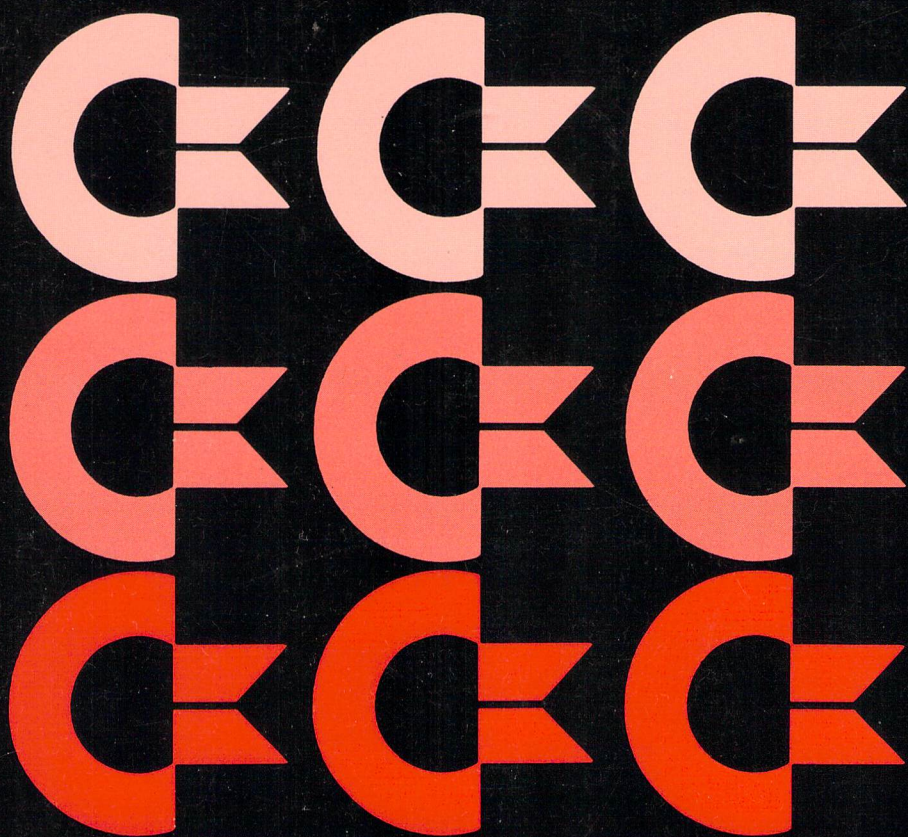


Shake hands with the

# Commodore 64



Pam Kelly-Hartley

Joy McKneil

Pitman



# Shake hands with the Commodore 64

## **Pam Kelly-Hartley**

T Dip T, T Dip PS, Dip Teaching

Technical teacher

Seven Hills College of Technical and Further Education  
Queensland

## **Joy McKneil**

GradDipEd Admin, Dip Bus Studies,  
Dip Teaching, T Dip T, Dip IPSA

Senior instructor, Business and General Studies  
South Brisbane College of Technical and Further Education  
Queensland

First published 1984

**Pitman Publishing Pty Ltd  
(Incorporated in Victoria)**

**158 Bouverie Street  
Carlton  
Victoria 3053**

**Level 12  
Town Hall House  
452-462 Kent Street  
Sydney  
New South Wales 2000**

**9th Floor  
National Bank Building  
420 George Street  
Brisbane  
Queensland 4000**

© P Kelly-Hartley, J McKneil 1984

National Library of Australia  
Cataloguing in Publication data

Kelly-Hartley, Pam, 1984-  
Shake hands with the Commodore 64.

Includes index.

ISBN 0 85896 154 7.

1. Commodore 64 (Computer).

I. McKneil, Joy, 1935-

II. Title.

001.64'04

Printed in Australia  
by Riall Print Pty Ltd

Designed by Sue Veitch

Cover design by Sue Veitch

Associated companies

Pitman Publishing Ltd  
London

Copp Clark Pitman  
Toronto

Pitman Publishing New Zealand Ltd  
Wellington



# Contents

Preface vii

## **1 Introduction to the Commodore 64 1**

Brief explanation of terms 2

## **2 The first handshake 5**

Starting the system 5

A quick look at the keyboard 5

Keyboard practice 8

How to clear the screen 9

Some editing features or how to correct errors 9

Cursor movement 10

## **3 Immediate execution of commands (1) 11**

### **—PRINT**

PRINT command 11

Abbreviated PRINT 12

Multiple PRINT 13

Length of PRINT statement 13

Methods of display 13

Combining some PRINT instructions 14

Summary 14

## **4 Immediate execution of commands (2)**

### **—Mathematical symbols 15**

Symbols for mathematical functions 15

Using decimals 17

Rounding 17

Scientific notation 17

Mathematical functions 18

Relational operators 18

Logical operators 19

Combinations of symbols 19

Order of precedence 20

Getting the right answer every time 20

## **5 Assignment of data to variables—LET 22**

Pigeon-holes or storage locations or addresses in memory 22

Storing data in memory 22

Retrieving data from memory 23

Changing data in memory 23

Performing calculations with data in memory 23

Clearing data from memory 24

Numeric variables 24

String variables 24

Using numeric and string variables 25

## **6 Deferred execution of commands (1)**

### **— Program execution 26**

- Sample program 26
- LIST 27
- Inserting a statement 27
- Changing a statement 28
- Deleting a statement 28
- Clearing the screen 29
- Clearing the memory 29
- Readability 29

## **7 Deferred execution of commands (2)**

### **— Simple programs 30**

- Using the printer 31

## **8 Disk drive and disks 34**

- Disk drive 34
- Disks 34
- Inserting a disk 36
- Formatting a disk 36
- Directory of disk 37
- Saving files on a disk 37
- Replace a file on disk 37
- Loading files from a disk 38
- Verify file on disk 38
- Copying a file on disk 38
- Renaming a file on disk 39
- Deleting a file from disk 39
- Reorganising the disk 39
- Disk protection 40
- To print a file saved on disk 40

## **9 Cassette recorder and cassettes 41**

- Saving files on cassette 41
- Verify file 41
- Loading a file from cassette 42
- Running the program 42
- Printing a file saved on cassette 42

## **10 Assignment of data to variables— INPUT READ DATA 43**

- Assigning data to variables 43
- INPUT statement 43
- READ and DATA statements 45

## **11 Loops—GOTO IF...THEN FOR and NEXT 48**

GOTO statement 48  
IF...THEN statement 50  
FOR and NEXT statements 52  
Variables — real or integer? 55

## **12 Loops—multiple and nested 56**

Multiple loops 56  
Nested loops 57

## **13 How to prepare a program 60**

Steps to follow 60  
Guidelines 61

## **14 More statements 71**

RESTORE statement 71  
Subroutine 71  
Dummy data 72  
LEN (length) statement 72  
ON...GOTO statement 72  
ON...GOSUB statement 73  
INT (integer) statement 73  
LEFT\$, RIGHT\$, MID\$ statements 74  
VAL (value) statement 74  
Writing your own program 74

## **15 Arrays 81**

One dimensional array 81  
Two dimensional array 82  
Nested loop 83

## **16 Colour and graphics 86**

Sprites 86  
How to enter programs in this section 87  
Colour 88  
Simple graphics 89  
Using control characters 89  
Programming screen positions 90  
Line drawing 91  
Picture drawing 92  
Screen and colour memory 93  
More difficult graphics 95  
Demonstration program 102  
Books and suggested reading 104

## **17 Sound 105**

- Programming music 106
- Programs—example 1 110
- Programs—example 2 111
- Demonstration program 112

## **18 Textfiles 117**

- What is a textfile? 117
- Two types of textfiles 118
- Method of using textfiles 119
- Sequential files 120
- Deleting a textfile SCRATCH 122
- Using keyboard to input and retrieve data 123
- Relative files 125

## **19 Word processing 128**

- Introduction 128
- What is word processing? 128
- Stages of word processing 128
- Starting the system 129
- Start-up options 129
- Formatting a document disk 130
- Learning about the options 130
- Saving on disk 132
- Direct cursor moves (non-destructive cursor moves) 133
- Viewing the directory 134
- Loading or retrieving a document from disk 134
- Standard or assumed format for printing 135
- Formatting a document 135
- Videoprint 137
- Scrolling — to move cursor 138
- Printing the directory 140
- Editing 140
- Printing a document 144
- Scanning the directory to load a document 144
- Deleting or scratching a document from disk 145
- Ranging 146
- Using the tabulator 147
- Widening text screen 149
- Changing video/screen width 149
- Decimal tabs 150
- Renaming a document 151
- Searching and replacing 151
- Hunting 152
- Standard paragraphs 153
- Headers and footers 155
- Indicating a new page (ie forced page) 156
- Automatic page numbering 156
- Additional functions 157
- Conclusion 158

## **Index 159**



# Preface

We are involved in teaching computer operation to post secondary students and find that available manuals assume an understanding of computing terminology. We realise that there is a need for a simplified guide for the beginner who has little or no knowledge of computers or computing terms. *Shake hands with the Commodore 64* has been designed to meet that need.

We wish to thank Software 80 of Taringa, Queensland, who provided the equipment and advice which made possible the completion of this book. We also wish to express our appreciation to those who have given us encouragement, advice and guidance. In particular we wish to acknowledge the contribution of Greg Perry, CCUG, Queensland, who wrote chapters 16 and 17 of the book.

Pam and Joy  
October 1984



# 1

## Introduction to the Commodore 64

*Shake hands with the Commodore 64* has been designed as an individualised program to help you learn to operate the Commodore 64 microcomputer.

All operating instructions relate specifically to the Commodore 64 microcomputer. Some of these operating instructions may also apply to other versions of Commodore microcomputers, but may not apply to other brands of microcomputers.

Knowing how the Commodore 64 operates will help you to understand how other microcomputers operate.



**Figure 1** The keyboard, monitor and cassette recorder of a Commodore 64 microcomputer.

All computer systems consist of:

- Hardware: the physical components of the computer
- Software: the programs that enable the hardware to be used
- Firmware: the programs permanently stored in read only memory.

## Brief explanation of terms

### Inside the Commodore 64

Inside the Commodore 64 are a number of parts which enable the microcomputer to operate. Some of these parts are:

#### 6510 Processor

- the central processing unit (CPU)
- contains special machine-language programs, which enable the CPU to collect information, follow instructions and produce results.

#### Memory

Computer memory is measured in **bytes**. One byte can contain one character (or similar amount of data).

There are 1024 **bytes** in one **kilobyte** and the capacity of computer memory is normally expressed in kilobytes (K).

eg 16K of memory =  $1024 \times 16 = 16\,384$  bytes

64K of memory =  $1024 \times 64 = 65\,536$  bytes

**Read-only memory (ROM)** contains 20 kilobytes of permanent programs to understand and respond to the instructions given by the operator. Read-only memory contains the BASIC language interpreter. This memory is constant, ie it never changes.

**Random-access memory (RAM)**. Data and/or instructions entered at the keyboard or obtained from disk are temporarily retained in random-access memory (RAM) to enable the current task to be performed. This memory is not constant, ie it retains data and/or instructions until these are replaced or until the computer is switched off. All details stored in RAM are lost when the power supply to the computer is turned off.

The Commodore 64 contains 64 kilobytes of random-access memory.

#### Circuit board with integrated circuits:

- a** the main circuit board
- b** special circuit boards

These allow the components to be connected to each other by means of specially shaped copper tracks.



## Display screen

The screen will display:

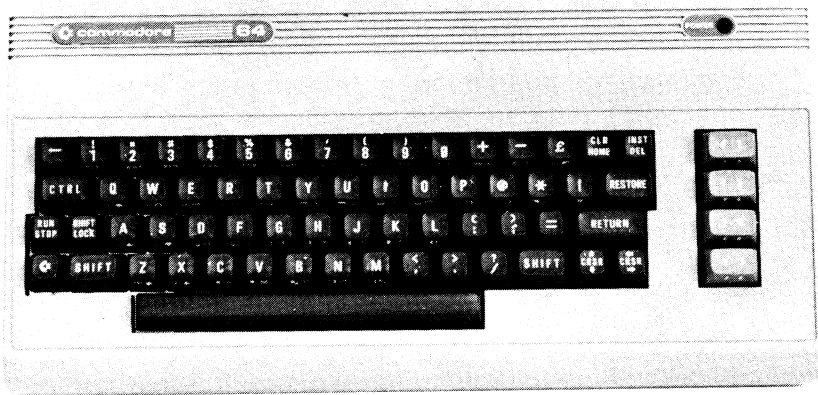
- text, ie capital letters A to Z, lower case letters a to z, figures 0 to 9 and special symbols.
- graphics characters
- special characters

The screen display allows the operator to see data and instructions entered through the keyboard or obtained from disk or cassette, and the results of those instructions.

## Keyboard

You will communicate with the Commodore 64 by using a keyboard very similar to a typewriter keyboard.

The keyboard is displayed in figure 2. Study it carefully and refer to it when new keys are introduced.



**Figure 2** The keyboard of a Commodore 64 microcomputer.

## Disk drive

The disk drive enables disks to be used to transfer instructions and/or data to and from the RAM and to provide a means of storing programs developed on the computer.

The operation of the disk drive can be compared to that of a cassette recorder. When the disk drive door is closed (similar to depressing the 'Play' button on a cassette recorder) the contents of the disk may be read.

## Speaker

An inbuilt speaker enables the Commodore 64 to produce sounds, which include a beep as a signal to the operator, and musical tones.

## **Disks**

A disk (or diskette) is a circular vinyl disk protectively enclosed in a flexible plastic envelope. A slot in the envelope allows access for the head of the disk drive to read the contents of the disk, while a notch on the side of the envelope will determine whether data may be written to the disk. If this notch is covered, the disk is 'write protected', ie new data cannot be recorded on the disk and so the data already on the disk will not be destroyed.

## **Cassette recorder**

You must use a specific (ie manufacturer's) cassette recorder to transfer programs and data on cassette tape to and from the RAM.

## **Cassettes**

Audio cassettes may be used for storage of programs, for loading pre-recorded programs into the computer and for storing your own programs.

## **Printer**

The printer is used to produce a printed copy of output from the microcomputer.

There are a number of printers which may be connected to the Commodore 64. The correct size and type of paper must be available for the type of printer used.

## **Languages**

The Commodore 64 can operate with a number of computer languages. One of the easiest computer languages for beginners to learn is known as BASIC.

BASIC is an acronym for:

**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode

All instructions in this book relate to BASIC language.

## 2

# The first handshake

In this section, you do not use disks or cassettes.

You will be working under the directions of the programs permanently stored in the ROM, and all data and/or instructions entered at the keyboard will be temporarily stored in RAM.

## Starting the system

- 1 Turn the system on by the power switch located on the right side panel.
- 2 Turn on the display screen.
- 3 The screen will now display two symbols — a prompt **READY** and a flashing cursor (■).  
The cursor (■) indicates the typing position on the screen.

## A quick look at the keyboard

The keyboard is very similar to the keyboard of an ordinary typewriter, but there are important differences.

### Touch

Touch will be slightly different from that of an electric typewriter and totally different from a manual typewriter.

### Text and graphic modes

There are two modes for keyboard entry:

- uppercase/graphic mode
- uppercase/lowercase mode

Each time the Commodore 64 is turned on, it will automatically be in the uppercase/graphic mode.

To move between the modes, depress and hold C= and then press SHIFT.

## Uppercase/graphic mode

Remember — when the Commodore 64 is turned on, it will always be in the uppercase/graphic mode. This will produce:

- with any letter key — uppercase letters, ie A–Z
- with any other key — the number/symbol shown on the bottom portion of the key, eg 2 8 , / ;
- with SHIFT and a non-letter key — the symbol shown on the top portion of the key, eg ! \$ # ?

Example: Depress and hold SHIFT; press 5. These two actions are abbreviated as Shift 5 and produce the symbol %.

- with SHIFT and a key with a graphic symbol — the symbol shown on the front right of the key.

## Uppercase/lowercase mode

Depress C= and SHIFT to move to uppercase/lowercase mode, which will produce:

- with any letter key — lowercase letters, ie a–z
- with any other key — the number/symbol shown on the bottom portion of the key.
- with SHIFT and a letter key — uppercase letters ie A–Z
- with SHIFT and a non-letter key — the symbol shown on the top portion of the key.
- with C= and a key with a graphic symbol — the symbol shown on the front left of the key.

## Shift lock

SHIFT LOCK is located near the bottom left of the keyboard.

Depress SHIFT LOCK until the key clicks into the ON (down) position to produce:

- in uppercase/graphic mode  
The graphic symbol shown on the front right of the key.  
The symbol shown on the top portion of a key.
- in uppercase/lowercase mode  
All uppercase letters  
The symbol shown on the top portion of a key.

To unlock the SHIFT LOCK, simply depress that key again until it clicks into the OFF (up) position.



## Figure 0

Note the difference between uppercase letter O and figure zero, ie 0. If you type a number containing a zero, you must use the figure 0.

## Figure 1

The figure 1 must be used for numerical work. Some typists use lower case L to indicate a 1 — the computer will not accept this.

## Special keys and their uses

RESTORE	This is the 'reset' key. When used in conjunction with RUN/STOP it will reset the computer as if it were just turned on. Any program in memory is still retained.
RETURN	Normally used to indicate to the computer that you have finished typing an instruction. Also moves the cursor to the beginning of the next line.
CONTROL	Used in conjunction with another key to perform additional functions. Used with a number key to produce colours. Can also be used to slow down a program LIST or RUN. (More about these later.)
COMMODORE C	Used before or with another key to perform additional functions. (More about these later.) Used with a number key to produce colours.
CRSR	Allows movement of cursor right or left. Press and hold for repeat action.
→RIGHT	Depress CRSR Moves the cursor forward one or more spaces.
←LEFT	Depress SHIFT CRSR Moves the cursor back one or more spaces.
CRSR	Allows movement of cursor up or down. Press and hold for repeat action.
↓DOWN	Moves the cursor down one line, without affecting typing.
↑UP	Depress SHIFT CRSR Moves the cursor up one line without affecting typing.
SPACE BAR	Enters one blank space and moves the cursor one space to the right. When space bar is used to space over previous typing, it removes those characters from the screen and from the computer memory.

INST/DEL	<p>To delete character to left of cursor, press INST/DEL.</p> <p>Press and hold for repeat action.</p> <p>To insert character — move cursor left, press SHIFT-INST/DEL, which provides space for insertion. Repeated action will allow more than one space. Type text to be inserted. Press RETURN.</p>
CLR/HOME	<p>CLR/HOME moves cursor to HOME position — upper left-hand corner of screen.</p> <p>SHIFT CLR/HOME clears the screen and moves cursor to HOME position.</p>
RUN/STOP	<p>RUN/STOP will stop program execution.</p> <p>SHIFT RUN/STOP will load a program from cassette tape.</p>
RVS ON/OFF	<p>Used to reverse the images produced on the screen.</p> <p>CTRL-9 produces reversed image.</p> <p>CTRL-0 returns normal image.</p>
FUNCTION KEYS	<p>On right of keyboard.</p> <p>May be defined to handle specific functions.</p>

## Keyboard practice

Type this section to practise keyboarding until you become familiar with the different touch required. Read the paragraphs before you commence typing.

- 1 Select the upper/lowercase mode.
- 2 For your keyboarding practice, you do not need to use RETURN. As you type, the cursor moves across the screen and then continues on to the next line, ie the cursor 'wraps around'.  
If you accidentally press RETURN during your keyboard practice, the message SYNTAX ERROR will appear on the screen. Ignore this message during your keyboard practice.
- 3 When the typing reaches the bottom of the screen and further characters are entered, the display gradually moves up the screen. This is called **vertical scrolling**.
- 4 If you make a typing error, use the cursor left to backspace to the incorrect letter, then type in the correction, then cursor right to move cursor back to original typing position.
- 5 The screen will display forty characters (numbered 0 to 39) across the screen and twenty-five lines (numbered 0 to 24) down the screen.

When the screen is full, try the next section.

## How to clear the screen

Depress `SHIFT CLR/HOME`. The screen is totally cleared and cursor has returned to top left of the screen, called the home position.

**Note:** RAM is not affected when the screen is cleared.

## Some editing features or how to correct errors

Type the previous section to improve your keyboard skills and practise these editing features.

- 1 `SHIFT CRSR-LEFT`** moves cursor to left to allow correction of typing error.
- 2 `CRSR-RIGHT`** moves cursor to right.  
Use `CRSR-LEFT` to position cursor over error. Type correction.  
Use `CRSR-RIGHT` to return to original typing position.  
If error is on previous line,  
use `CRSR-UP` to move cursor to position of error, type correction and press `RETURN`.
- 3 `DELETE`** press `INST/DEL` to delete character to left of cursor. Press and hold for repeat action.  
Now delete the last four words of your screen display.
- 3 `INSERT`** position cursor between two words of your screen display.  
press `SHIFT-INST/DEL` which will provide one space to allow insertion of one letter.  
press and hold `SHIFT-INST/DEL` to provide more than one space for insertions.  
Now insert sufficient text to fill the spaces and then press `RETURN`.

**Note:** In `INSERT` mode, using any cursor key will produce a different (strange) screen symbol, which is actually programmed cursor mode. These symbols are used within a program and will be explained in a later chapter. To escape this programmed cursor mode, simply press `RETURN`.

# Cursor movement

When the cursor reaches:

- the top of the screen — it will stop.
- the bottom of the screen — the cursor remains stationary, but the printing on the screen moves up.
- the right edge of the screen, with further moves right — it will reappear at the left margin one line lower.
- the left margin of the screen, with further moves left — it will reappear on the right edge, one line higher.



# 3

## Immediate execution of commands (1) PRINT

The functions explained in this chapter are available under the directions of the programs permanently stored in ROM. All data and/or instructions entered at the keyboard will be temporarily stored in RAM. You will not need to use either a disk or cassette.

When you were practising your keyboarding skills, you did not have to use the RETURN key at the end of each line. If you accidentally pressed RETURN, the message SYNTAX ERROR appeared on the screen, which you were told to ignore.

In fact, the RETURN key has a special function — it is a signal to the computer that you have finished typing an instruction. In *immediate execution* the computer must now obey your instructions.

A word of warning though. If you type an instruction and press RETURN, and the message SYNTAX ERROR appears, the computer is telling you: 'You have made an error — I cannot read your instruction'. If this happens, check your instruction carefully and then retype it correctly.

If you notice a typing error before you press RETURN, use the editing keys to correct the error.

Call up prompt and cursor.

**Note:** In uppercase/graphic mode, commands (eg PRINT) must be typed in uppercase.

In uppercase/lowercase mode, commands (eg print) must be typed in lowercase.

### PRINT command

Use uppercase/graphic mode.

- 1 Type PRINT "HELLO" then press RETURN.
- 2 The computer has followed your instructions — it has printed the word HELLO

**Note:** Any characters enclosed in quotation marks (string data) will be printed exactly by the PRINT command. The first quotation mark indicates that the following character is the start of the material to be printed, while the second quotation mark indicates that the material to be printed has been completed.

**3** Type PRINT "MY NAME IS . . ." then press RETURN

**4** Try printing some more words and short sentences, remembering to press RETURN when you have finished typing your instruction.

You should understand the function of the RETURN key by now, so the instruction — press RETURN — will not be shown in the following instructions.

**5** What happens if you forget to type the first quotation mark? Try it.

Type PRINT YOUR NAME"

The result: Ø The computer could not read your command as there was no opening instruction.

**6** Type PIRNT "YOUR NAME" deliberately misspelling PRINT to see the computer's reaction.

The result: SYNTAX ERROR The computer could not read your instruction.

**7** Type PRINT "1234" Result?

**8** Type PRINT 1234 Result?

**Note:** Numbers (numeric data) will be printed exactly whether enclosed or not enclosed in quotation marks.

**9** Type PRINT "30 JUNE 1984" Result?

**10** Type PRINT "30/6/1984" Result?

**11** Type PRINT 30/6/1984 What happened?

The computer reads a / as meaning 'divide by', so in this example 30 was divided by 6, giving 5 and 5 was divided by 1984.

**Note:** To PRINT an exact copy of a mixture of numerics and non-numerics (string data), all characters must be enclosed in quotation marks.

Practise this section until you feel confident about printing an exact copy of any characters.

## Abbreviated PRINT

When you are using BASIC, the PRINT command can be shortened to a question mark.

PRINT "JANUARY"

? "JANUARY" Was the result the same?

From now on, you may use either the word PRINT or the question mark.

## Multiple PRINT

You may use the PRINT command more than once in the same statement. Try this example.

```
PRINT "MONDAY": PRINT "TUESDAY"      Result?
```

**Note:** The colon is used in this example to indicate a following PRINT.

## Length of PRINT statement

You can instruct the computer to print a longer message. However, there is a limit to the length of the message. If the message contains too many characters, it will not be printed, even though the computer has accepted the input.

In fact, after pressing RETURN, you will get a SYNTAX ERROR message to let you know that the computer is unable to follow your instruction.

Try this yourself, to see the computer's reaction. Use the first paragraph of this section as your message. Remember not to use the RETURN key to continue typing on the next line.

```
PRINT "YOU CAN INSTRUCT THE COMPUTER ..."
```

## Methods of display

There may be occasions when you want to space your work across the screen, or close-print your work. Try the following methods.

### 1 PRINT "HEADING", "HEADING", "HEADING"

The comma instructed the computer to print the words in columns. Columns are preset every ten spaces across the screen.

### 2 PRINT "MONDAY"; "TUESDAY"; "WEDNESDAY"

The semicolon instructed the computer to print the word in the next available space, so that the words run together. How could you get a space between the words so that they do not run together?

Try PRINT "MONDAY "; "TUESDAY "; "WEDNESDAY"

By leaving a space between the last letter of each word and the quotation mark, the printing will also leave a space.

### 3 PRINT TAB(5) "MONDAY" TAB(20) "TUESDAY"

The word TAB instructed the computer to print the words across the screen at the character position on the line indicated by the figure enclosed in brackets.

Practise these methods with some examples of your own.

## Combining some PRINT instructions

So far, you have found out that:

- **PRINT** or **?** are commands to the computer to print your message.
- The comma is an instruction to print in columns.
- The semicolon is an instruction to close-print.
- The colon allows more than one **PRINT** command in a statement.
- The **TAB** command instructs printing at a specified line position.

Let's try combining some of these instructions.

- 1 **PRINT** "EXAMPLES", "OF", "DISPLAY"
- 2 **PRINT** **TAB**(5) "EXAMPLES", "OF", "DISPLAY"
- 3 **PRINT** **TAB**(5) "EXAMPLES" **TAB**(20) "OF" **TAB**(30) "DISPLAY"
- 4 **PRINT** "EXAMPLES" :?"OF" :?"DISPLAY"

Try some of your own combinations and also some longer messages. Think carefully about the display you want, and make sure that you give the computer the correct instructions.

## Summary

- Any characters enclosed in quotation marks (string data) will be printed exactly.
- Numbers (numeric data) will be printed exactly whether enclosed or not enclosed in quotation marks.
- A mixture of numerics and non-numerics (string data) must be enclosed in quotation marks.
- The **PRINT** command can be abbreviated to a question mark.
- The **PRINT** command may be used more than once in the same statement by using a colon.
- The length of a **PRINT** statement is limited.
- The comma is an instruction to print in columns.
- The semicolon is an instruction to close-print.
- The **TAB** command instructs printing at a specified line position.

## 4

# Immediate execution of commands (2) Mathematical symbols

By the time you start this chapter, you already know that you must type **PRINT** or **?** to command the computer to print your message. So from now on the command **PRINT** will not be shown.

Remember that you are still working under the direction of permanent programs in **ROM**. Keyboard entries are temporarily stored in **RAM**.

Call up prompt and cursor.

## Symbols for mathematical functions

The computer can be used to supply answers to mathematical problems. Use the following symbols:

- + Addition
- − Subtraction
- \* Multiplication
- / Division
- ↑ Exponentiation (eg squaring)
- ↑ (1/2) Square root of . . . (or exponentiation to the 1/2 power)

When you are typing the following problems, you can:

- Leave a space between characters, eg  $4 + 4$
- Leave no space between characters, eg  $4+4$

Either way, the computer will read your instructions.

### Addition

**1**  $4 + 4$

**2**  $75 + 17$

**Note:** The computer will supply only the answer. To make the result more meaningful, type

"75 + 17 = ";75+17      Result: 75 + 17 = 92

If you use a comma instead of a semicolon after the closing quotation mark, the answer will be printed in the next column.

**3** "110 + 78 + 97 = ", 110+78+97

What happens if you leave out the first quotation mark? Try it and see.

The message you received tells you that the computer cannot understand your instruction.

**4** "4 + 6 = ";4+6

What happens if you enter the wrong figures? Try this.

"4 + 5 = ";4+6

Result 4 + 5 = 10 which you know is wrong!

But the computer followed your instructions by:

**a** printing the quoted figures 4 + 5 =

**b** adding the numbers 4+6 10

So you must be very careful in entering data — always check your typing before pressing RETURN.

*Try the following:*

## Subtraction

**1** 117 - 36

**2** "42 - 19 = ";42 - 19

**3** "176 - 28 - 43 = ";176-28-43

## Multiplication

**1** 42 \* 6

**2** "17 \* 31 = ";17\*31

**3** "82 \* 4 \* 12 = ";82\*4\*12

## Division

**1** 1776/12

**2** "273/13 = ";273/13

**3** "3976/4/7 = ";3976/4/7

## Squaring or cubing a number

**1** 5 ↑ 2                      ie 5 \* 5

**2** "6 ↑ 3 = "; 6 ↑ 3    ie 6 \* 6 \* 6

## Finding the square root or cube root

- 1  $25 \uparrow (1/2)$  ie the square root of 25
- 2 It is much easier to use the SQR function  
eg SQR(49) SQR(23104)
- 3  $216 \uparrow (1/3)$  ie cube root of 216

## Using decimals

Decimals may be used in all calculations. There are some points you should remember though.

- If you add 10.14 and 2.16 the answer is 12.30. The computer will print 12.3, ie it will not print trailing zeros.
- If you add 033.62 and 021.20 the answer is 054.82. The computer will print 54.82, ie it will not print leading zeros.

## Rounding

If you type a number with more than nine digits, or if the answer to a problem contains more than nine digits, the computer will round the number to nine digits.

### Example

Type PRINT 2345.67891234

Result: 2345.67891      Rounded down because the tenth digit was less than 5.

Type PRINT 2345.67898933

Result: 2345.67899      Rounded up because the tenth digit was equal to or greater than 5.

## Scientific notation

If you type in a long number, the result may be shown with an included letter E.

### Example

Type PRINT 1234567899876543      Result?

This is called scientific notation.

Follow these examples:

- 1 Type PRINT 0.00356      This will display as 3.56E-03

To get the correct answer, look at the symbols following the character E.

The minus symbol (-) means 'move the decimal point to the left'; the numbers (03) mean '03 spaces'. So the correct figure is 0.00356.

**2** Type PRINT -9876543210 This will display as  
-9.87654321E+09

The plus symbol (+) means 'move the decimal point to the right'; the numbers (09) mean '09 spaces'. So the correct answer would be  
-9876543210.

Unless you understand scientific notation, use simpler calculations until you feel confident with the system.

## Mathematical functions

There are many mathematical functions available and if you intend to use the computer for more complex mathematical problems, you should check the User Guide for a complete listing.

Try these three examples.

**1** Square root

SQR(169)

**2** Integer (whole number)

Returns largest whole number less than number

INT(16.7)

INT(16.2)

**Note:** By adding .5 to the number in brackets, the **integer** function can be used to round numbers (up or down) to the nearest whole number.

INT(16.7+.5)

INT(16.2+.5)

**3** Random number

Returns a different number each time the function RND is used. Work each example at least twice

RND(1)

RND(5)

## Relational operators

Relational operators can be used to compare data and to determine if the data is the same value, or a different value.

Use the following symbols:

< less than

> greater than

= equal to

<> not equal to

<= less than or equal to

>= greater than or equal to



Try these examples. Remember to use PRINT and RETURN.

1  $10 = 10$      Result   -1   means TRUE

2  $10 = 11$      Result   0   means FALSE

3  $10 > 8$

4  $98 <> 98$

5  $78 \leq 97$

6  $1983 \geq 1982$

7 "MONDAY" = "MON"

8 "THEIR" = "THERE"

## Logical operators

Logical operators test the comparison between two or more relations. The logical operators are AND OR NOT .

Do not use PRINT in these examples.

AND IF  $1 = 1$  AND  $2 = 2$  THEN PRINT "TRUE"

If both statements are true, the screen will display TRUE .

If either statement is incorrect, nothing will be displayed, eg IF  $1 = 1$  AND  $1 = 2$  THEN PRINT "TRUE" would result in nothing displayed.

OR IF  $2 = 2$  OR  $3 = 2$  THEN PRINT "TRUE"

If either statement is correct, TRUE will appear on the screen.

If neither is correct, nothing will be displayed, eg IF  $2 = 3$  OR  $3 = 4$  THEN PRINT "TRUE" would result in nothing displayed.

NOT IF NOT  $(7 = 7)$  THEN PRINT "TRUE"

If 7 is not 7 TRUE will appear on the screen.

For a full understanding of logical operators, work through these on the computer.

## Combinations of symbols

*Addition and subtraction*

$202 - 57 + 143 - 98$

Calculations are performed from left to right.

*Addition, subtraction and multiplication*

$202 + 57 - 25 * 2$

The multiplication is performed first.

Then additions and subtractions are performed from left to right.

*Addition, subtraction, multiplication and division*

$987 + 17 - 30 * 2 + 88/11$

The multiplications and divisions are performed first, from left to right.

Then additions and subtractions are performed from left to right.

*Addition, subtraction and multiplication, division and exponentiation.*

$$10 \uparrow 3 + 1079 + 10 * 6 - 31/16 - 4 \uparrow 2$$

Exponentiation (eg squaring) is performed first, from left to right.

Then multiplications and divisions are performed from left to right.

Then additions and subtractions are performed from left to right.

The computer has definite rules for performing calculations. These rules are called **order of precedence**.

## Order of precedence

Order of precedence determines the order of the steps for all mathematical calculations.

Step 1 ( ) Any calculations enclosed in brackets, from left to right.

Step 2 - Negative numbers

Step 3  $\uparrow$  Exponentiation from left to right.

Step 4 \*/ Multiplication and division from left to right.

Step 5 + - Addition and subtraction from left to right.

**Rule:** If there is more than one calculation using the same symbol, calculations are performed from left to right.

Study this example and follow the steps:

Example  $-2 + 20 + 120 + 6/3 - 4 * (2 + 10) + 6 \uparrow 2$

Step 1: brackets  $-2 + 20 + 120 + 6/3 - 4 * (2 + 10) + 6 \uparrow 2$

Step 2: negative number  $-2 + 20 + 120 + 6/3 - 4 * 12 + 6 \uparrow 2$

Step 3: exponentiation  $18 + 120 + 6/3 - 4 * 12 + 6 \uparrow 2$

Step 4: multiplication and division  $18 + 120 + 6/3 - 4 * 12 + 36$

Step 5: addition and subtraction  $18 + 120 + 2 - 48 + 36$

## Getting the right answer every time

Work these samples yourself (without the computer) and then check your answers on the computer.

$$6 * 6 + 4 * 6 \uparrow 2 + 10$$

$$10 * 4 * -2 + 10 * 5 \uparrow 3$$

Notice the different results in the next two problems because of the inclusion of brackets.

$$10 + 2 * 4 - 6/3 = \quad (16)$$

$$(10 + 2) * (4 - 6/3) = \quad (24)$$

Now work some examples of your own (without the computer) and then check your answers on the computer.

# 5

## Assignment of data to variables

### LET

Call up prompt and cursor.

**Remember:**

- ROM contains the programs to process your instructions.
- RAM contains the data and/or instructions entered via the keyboard.

### Pigeon-holes or storage locations or addresses in memory

Every computer has the capability of storing data for later use. Using the BASIC language, there are hundreds of pigeon-holes into which data can be placed and recalled later. You, as the operator, can name each pigeon-hole so that you will know where to find any data.

In this section, you will use the letters A B C D E F as names for six pigeon-holes.

Use uppercase/graphic mode.

### Storing data in memory

This means inserting a specific data item into the named storage location in memory.

Do not use PRINT. The word LET must be typed in uppercase.

Type   LET A = 344  
          LET B = 125  
          LET C = 22

The word LET is optional. Type the following data without using LET.

D = 5

E = 55

F = 4

Each number has now been placed in its own pigeon-hole.

## Retrieving data from memory

Instruct the computer to print the contents of each pigeon-hole.

PRINT A, B, C

PRINT D, E, F

Are they all correct?

## Changing data in memory

Type A = 255

PRINT A

What happened? The previous data (344) has been removed from the pigeon-hole and cancelled from memory, and has been replaced by the new data (255).

## Performing calculations with data in memory

Instruct the computer to carry out these tasks. Remember to use the commands PRINT and RETURN, and use the mathematical symbols you have already learned.

- 1 A plus B plus C
- 2 E minus D minus F
- 3 A multiplied by E multiplied by C
- 4 B divided by D
- 5 A plus D multiplied by F
- 6 A plus B divided by D
- 7 D squared
- 8 Square root of B
- 9 C multiplied by (E plus A)
- 10 A minus (F multiplied by D) plus (B plus C)

After doing the calculations, check the contents of the pigeon-holes again. They should be the same as before.

PRINT A, B, C

PRINT D, E, F

Try some calculations of your own, using the data already stored in the pigeon-holes.

## Clearing data from memory

When you have finished your calculations, you can clear all the pigeon-holes and set them to zero by typing `NEW`.

Now try `PRINT A, B, C`

## Numeric variables

The correct name for pigeon-holes is **variables**. This is the term that will be used from now on.

In the previous examples, you used a letter to identify each variable. All the data you used was numeric, ie numbers only. When the data is all numeric, you must use a numeric variable.

### Numeric variables

#### 1 Integer (ie whole number)

- must begin with a letter (A-Z)
- may be followed by another letter (A-Z) or a digit (1-9)
- must end with a percent sign (%)

eg `A% = 10`  
`LZ% = 20`  
`J2% = 30`

#### 2 Real (ie number with a decimal portion)

- must begin with a letter (A-Z)
- may be followed by another letter (A-Z) or a digit (0-9)

eg `A = 10.27`  
`LZ = 21.96`  
`J2 = 37.02`

## String variables

What happens if you want to store data that is not all numeric?

Data that is not all numeric is called string data. It consists of a string of letters (alphabetic) or a mixture of letters, numbers and symbols (alphanumeric).

String data must use a string variable and must be enclosed in quotation marks.

## String variables

- must begin with a letter (A-Z)
- may be followed by another letter (A-Z) or a digit (1-9)
- must end with a dollar sign (\$).

**Note:** When selecting letters to use as a variable (either numeric or string) do not use the combinations of TI or ST as these are reserved for special functions.

## Using numeric and string variables

Try these examples:

```
1 Type  A$ = "STRING DATA  "
        B$ = "IN QUOTATION MARKS"
        PRINT A$; B$
```

Did you notice that there is a space between DATA and the closing quotation mark?

```
2 Type  A% = 11
        B = 14.5
        M$ = "MARY IS  "
        R$ = "ROGER IS  "
        Y$ = "  YEARS OLD"
        PRINT M$; B; Y$
        PRINT R$; A%; Y$
```

Try some examples of your own, combining numeric and string variables. When you have finished, remember to clear all variables.

# 6

## Deferred execution of commands (1) Program execution

**Deferred execution** simply means that you are going to input a number of instructions, to be performed later. This is really writing a simple computer program.

The major difference between a program written for deferred execution and statements written for immediate execution is that in deferred execution each statement must be preceded by a number.

Call up prompt and cursor.

### Sample program

- 1 Input as shown, inserting your own personal details in statements 30, 40, 50 and 60.

```
10 REM MY FIRST PROGRAM
20 PRINT "EXERCISE 1"
30 PRINT "YOUR NAME"
40 PRINT "NUMBER AND STREET"
50 PRINT "SUBURB"
60 PRINT "PHONE NUMBER"
9999 END
```

LIST (watch computer reaction to this command)

RUN (watch computer reaction to this command)

- 2 What happened?

The computer followed your instructions by:

- a storing your statements (numbered 10 to 9999) in RAM
- b Listing your statements for you to check
- c running your instructions under the directions of programs stored in ROM.



**3** Let's look closely at each part of your program.

## **NEW**

**NEW** is an instruction to the computer to wipe out previous data stored in memory (ie clear all variables) and to store a brand new program.

## **Statement numbers, eg 10 to 9999**

The computer stores statements in ascending order, ie lowest to highest. It is always a good idea to increase statement numbers by 10 (as you did), in case you have to insert extra statements when you are checking the program. The highest statement number you can use is 63999.

## **REM**

**REM** is short for remark. **REM** statements are for reference purposes only — the computer does not act on them. They help to identify and clarify your program.

## **END**

**END** signifies to the computer the end of the program. For your exercises, use 9999 to signify **END**.

## **LIST**

**LIST** is an instruction to the computer to list your program on the screen so that you can check it.

## **RUN**

**RUN** is an instruction to the computer to obey your program instructions.

## **LIST**

When you type **LIST**, the whole program is listed on the screen. It is also possible to list portions of the program.

**LIST**                      Displays whole program

**LIST 40**                Statement 40 is displayed

**LIST 30–50**          Statements from 30 to 50 are displayed.

**LIST –40**            Statements from beginning to statement 40 are displayed.

**LIST 40–**            Statements from statement 40 to **END** are displayed.

Remember — the **CONTROL** key can be used to slow down a listing.

## **Inserting a statement**

Your program (exercise 1) is still stored in computer memory. Type **LIST** to display the program on the screen.

Now you are going to insert three additional statements in your program. Two statements will be to insert a blank line after “EXERCISE 1” and after “YOUR NAME”. The third insert will be your postcode, positioned after your suburb.

Type (at the cursor position)

```
25 PRINT
35 PRINT
55 PRINT "POSTCODE"
```

LIST

Notice how the computer has re-organised the lines in ascending statement number.

RUN

## Changing a statement

If you want to change the contents of a statement (or correct an error) you may use either of the following two methods.

*To renumber your program as exercise 2*

Method 1

Retype the entire statement:

```
10 REM MY SECOND PROGRAM
20 PRINT "EXERCISE 2"
```

Method 2 Cursor movements

- a** Type LIST
- b** Use SHIFT-CRSR up to position cursor on line containing error.
- c** Use CRSR-RIGHT to position cursor on error.
- d** Type correction, or insert/delete characters.
- e** Press RETURN to enter the change to memory and CRSR down to return cursor to end of program.
- f** Type LIST to check corrections.

## Deleting a statement

If you want to DELETE a portion of your program, eg to remove the exercise number

LIST or LIST 20

Then type:

20 (this will delete the previous statement 20)

LIST (to check deletion)

RUN

Now delete statements 40 to 55

LIST

RUN

## Clearing the screen

Clear the screen by SHIFT-CLEAR/HOME.

Is your program still in memory? Type LIST to find out.

## Clearing the memory

The only way to clear your program from memory is to type NEW .

Now try LIST (program has been wiped from memory)

## Readability

Did you notice that when a program is listed, the computer has inserted additional spaces to make it easier to read?

# 7

## Deferred execution of commands (2) Simple programs

Now you are going to input, LIST and RUN a new program, make alterations to that program and then LIST and RUN the amended program.

### Exercise 3

Enter this program.

```
10  REM  SIMPLE CALCULATIONS
20  PRINT "EXERCISE 3"
30  PRINT "*****": PRINT
40  PRINT "YOUR NAME": PRINT
50  A% = 50:B% = 10:C% = 12
60  PRINT "A% = ";A%: PRINT
70  PRINT "B% = ";B%: PRINT
80  PRINT "C% = ";C%: PRINT
90  PRINT "A% + B% * C% = ";A% + B% * C%
100 PRINT
110 PRINT "A%/B% * C% = ";A% / B% * C%
9999 END
```

LIST and RUN

If you leave out an instruction, eg PRINT, and then try to RUN the program, you will receive a syntax error message, identifying the statement by number. Check that statement number in your listed program and make the necessary correction. Try this yourself, by deliberately leaving out the word PRINT in statement 60.

Now make these alterations to your program:

Change A% to 100, B% to 20 and C% to 24.

Delete statements 60 to 80.

LIST and RUN

## Exercise 4

This program works out the cost of buying 48 articles at \$17.58 each. See if you can understand the steps, then enter the program.

```
10 REM TO DETERMINE COST
20 PRINT "EXERCISE 4"
30 PRINT "*****": PRINT
40 PRINT "YOUR NAME": PRINT
50 REM N% = NUMBER OF ARTICLES
60 REM P = PRICE OF EACH ARTICLE
70 REM C = TOTAL COST
80 PRINT "NUMBER","PRICE","COST $"
90 N% = 48:P = 17.58:C = N% * P
100 PRINT N%,P,C
9999 END
```

LIST and RUN.

## Exercise 5

Make the following changes to exercise 4:

Change to exercise 5.

Delete statements 40 and 50.

Insert a blank line after statement 90.

Change N% to 75

Change P to 25.99

LIST and RUN

## Exercise 6

Design a simple program to solve this problem.

If you purchased 32 articles, and the total cost was \$497.60, what was the price of each article?

LIST and RUN.

## Exercise 7

Design a simple program to find the area of a rectangle where L = length, W = width and A = area.

Your program should include a statement to print the following:

AREA OF RECTANGLE IS . . . . SQ CM.

LIST and RUN.

## Using the printer

Before proceeding, check that the printer has a supply of paper inserted. See the manual for your particular printer for details of paper insertion.

## To print a program list

- 1 Enter a program.
- 2 LIST that program to check accuracy.
- 3 Turn printer on.
- 4 The opening command for the printer is:  
OPEN filename, 4  
where you may select any number from 1 to 128 to indicate filename.  
The number 4 indicates the printer.  
Type OPEN 3, 4
- 5 The command to transfer control to the printer is:  
CMD filename  
where filename *must* be the same as in the OPEN command.  
Type CMD 3  
The printer will respond by printing READY.
- 6 Type LIST  
Program LIST will be printed.
- 7 The command to transfer control back to the computer is:  
PRINT #filename  
where filename *must* be the same as in the OPEN command.  
Type PRINT #3
- 8 To close the filename you have been using:  
CLOSE filename  
where filename *must* be the same as in the OPEN command.  
Type CLOSE 3

The complete process to produce a printed LIST of a program is:

<i>Program must be in memory</i>	<i>Example</i>
OPEN filename, 4	OPEN 3, 4
CMD filename	CMD 3
LIST	LIST
PRINT #filename	PRINT #3
CLOSE filename	CLOSE 3

## To print a program run and list

- 1** Enter a program.
- 2** LIST.
- 3** Insert two statements before the first statement:  
1 OPEN filename, 4  
2 CMD filename  
where you may select any number from 1 to 128 to indicate filename.
- 4** Insert a statement before the END statement:  
no. LIST
- 5** Type 1 OPEN 3, 4  
2 CMD 3  
100 LIST
- 6** Type LIST to check accuracy.
- 7** Now type RUN  
Program RUN will be printed, followed by program LIST.
- 8** When the printing is completed, type:  
PRINT#3  
CLOSE 3

The complete process to produce a printed RUN and LIST of a program is:

*Program must be in memory*

*Example*

Insert

1 OPEN filename, 4

2 CMD filename

no. LIST

RUN

PRINT#filename

CLOSE filename

1 OPEN 3, 4

2 CMD 3

100 LIST

RUN

PRINT#3

CLOSE 3

# 8

## Disk drive and disks

You may omit this chapter if you are using only a cassette recorder and cassettes.

Using a disk drive and disks is a convenient and efficient method to access a large amount of data and to provide secure storage for your own programs.

### Disk drive

The operation of the disk drive can be compared to that of a cassette recorder. Compare the following steps:

#### *Cassette Recorder*

Insert cassette

Close door

Depress PLAY button }

Head lowered on cassette

Sound is produced

#### *Disk Drive*

Insert disk

Close door

Head lowered towards disk (but not actually touching)

Contents are 'read'

### Disks

Disks are used for storage of data and retrieval of this data for future reference. The data is filed under filename, eg surname, subject, etc.



## Make-up

The disk you will be using will probably be a 'floppy' disk. This means that the disk is flexible — but bending will damage it. The protective cover contains both lubricants and cleaning agents. Always treat the disk with respect.

The disk is a small 12.5 cm plastic disk coated so that data may be magnetically stored on it and, if desired, erased from its surface. The coating is similar to that on a recording tape/cassette.

The disk is sealed in a square plastic cover to protect it. The cover helps to keep it clean and allows it to spin freely in the disk drive. **The cover is never opened.**

Some disks have a reinforced section around the centre spindle hole. This helps to prevent excessive wear on the inner edge of the disk.

There is a slot in the cover through which the 'head' is able to read the contents.

The disk can store over one million bits of data. Therefore each individual bit occupies a very small portion only, so an invisible scratch or even a fingerprint can cause an error.

## Care

Never let anything touch the brown/grey surface (especially fingers) — handle the plastic cover only.

## Average life

The average life of a disk is 40 hours. Considering the few seconds it takes to read the data, the disk should last a long time.

## Storage

Keep the disk in a paper jacket when not in use to minimise static electricity build-up which attracts dust. It is preferable to store the disk vertically.

Keep the disk out of the sun and away from heat. The disk can warp and the first evidence of heat damage is a warped or bent plastic cover.

## Label

Always label disks so you will know the contents. Write the title on a label and attach to the disk. Do not write on a label already attached to the disk, as the pressure of a pen could damage the disk.

## Write-protecting

You will notice a small 'cut-out' on the side of the cover — this may be covered with a tab so the recorded contents of the disk cannot be erased. This is very similar to a cassette where the two small lugs at the back are removed so that nothing can be recorded over and hence nothing can be

erased. A disk that you intend to work on should have this tab removed, so that information can be recorded on and read from the disk.

## Inserting a disk

- 1 The green power on light must be on.
- 2 The red in use light on the front of the disk drive must be off.
- 3 Hold the disk carefully in a horizontal position without touching the exposed sections. The label should be facing up. This means that the write-permit notch (or write-protect tab) will be to the left as the disk is inserted. You will feel the disk seat into place.
- 4 Close the drive door. This lowers the 'head' towards the disk so that it can 'read' the contents. However, the head does not actually touch the disk.
- 5 The red in use light on the drive will light up while the motor is working. The data and/or instructions on the disk will be transferred to RAM. **Do not attempt to use the machine while this red light is on**, ie do not open the disk drive door or use the keyboard.

**Note:** If the red light is flashing (which indicates an error), but the drive motor has stopped, the disk may be removed.  
Never turn the disk drive on or off when the disk drive door is closed and a disk is in place.

## Formatting a disk

A blank disk is of no use to your Commodore 64. It needs to be set out magnetically in much the same way as you might draw lines on a blank page before you start to write.

The term **formatting a disk** means:

- preparing a totally new disk for use, or
- deleting all the contents of a previously used disk, so that it may be re-used.

### To format a new disk

- 1 Insert disk in drive and close drive door.
- 2 Type OPEN 15, 8, 15 and press RETURN.
- 3 Type PRINT #15, "NEW0:diskname, id" and press RETURN.  
CLOSE 15

You may select any diskname which then becomes the disk name. You may also select any two characters to represent id.

Formatting will start immediately, and once the process is complete and the red in use light goes off, the disk is ready for use.

## To format a previously used disk

- 1 Follow instructions in **1** and **2** above.
- 2 Type `PRINT #15, "NEW0:diskname"` and press RETURN.  
`CLOSE 15`

**Notes:** The number 8 in the above commands indicates the disk drive. The number 0 in the above commands indicates drive 0. The command NEW may be abbreviated to N.

## Directory of disk

The directory keeps a record of the disk name and the number of blocks (space) available on the disk.

To view directory, type `LOAD "$", 8` and press RETURN.

The red in use light will come on and in a few seconds the screen will display SEARCHING FOR \$ and then LOADING and READY.

The directory is now loaded into memory.

Type LIST to display the directory on the screen.

## Saving files on a disk

At this point, you will learn how to save one of your simple programs on disk for future reference.

- 1 Insert the disk which you have just formatted.
- 2 Type `NEW`
- 3 Enter a simple program, one of your previous exercises.
- 4 `LIST` and `RUN` the program.
- 5 Select a filename. The filename may begin with a letter or a number and may contain up to 16 characters.
- 6 Type `SAVE "filename", 8` eg `SAVE "EXERCISE 1", 8`  
The data and/or instructions in your program will be copied from RAM to the disk.

When you have saved the program, check the directory by

`LOAD "$", 8` and press RETURN

`LIST`

The program you have saved will now be shown in the directory.

## Replace a file on disk

If you have a program already saved on disk and at a later date you wish to

change and/or extend that program, you can use the **SAVE** and **REPLACE** command.

**SAVE**“@0:filename”, 8

## Loading files from a disk

Now that you have a program stored on disk, it may be recalled at any time by:

**LOAD** “filename”, 8      eg **LOAD** “EXERCISE 1”, 8

**LIST**      for screen listing

**RUN**      for program execution.

## Verify file on disk

When you have saved a program on disk, you may wish to verify it by comparison with the program still in memory.

Enter a short program (exercise 2) and then **LIST**, **RUN** and **SAVE** to disk. Now verify by:

**VERIFY** “filename”, 8      eg **VERIFY** “EXERCISE 2”, 8  
or **VERIFY** “\*”, 8      where \* indicates last used program name

If the program is correctly saved, the screen will display **SEARCHING FOR filename, VERIFYING** and then **OK** and **READY**. If not correctly saved, an error message will be displayed.

## Copying a file on disk

You may want to keep two copies of the same program under different filenames. You may use either of these two methods:

### Method 1

- **LOAD** a program from disk to memory.
- **LIST** and **RUN** to check accuracy.
- **SAVE** using another filename.

eg **LOAD** “EXERCISE 1”, 8  
**LIST**  
**RUN**  
**SAVE** “EXERCISE 6”, 8

- Check directory.

### Method 2

The **COPY** command will store on disk the contents of a file under a different name.

```
OPEN 15, 8, 15
PRINT#15, "COPY0:newfile=0:oldfile"
CLOSE 15
```

eg to copy exercise 2 as exercise 3

```
OPEN 15, 8, 15
PRINT#15, "COPY0:EXERCISE 3 = 0:EXERCISE 2"
CLOSE 15
```

Check the directory.

## Renaming a file on disk

There may be a reason for you to rename a file which has already been saved.

```
OPEN 15, 8, 15
PRINT#15, "RENAME0:newname=oldname"
CLOSE 15
```

Now rename your EXERCISE 1 as EXERCISE 4.

```
OPEN 15, 8, 15
PRINT#15, "RENAME0:EXERCISE 4= EXERCISE 1"
CLOSE 15
```

Check the contents of the directory.

## Deleting a file from disk

If you wish to delete a file from the disk:

```
OPEN 15, 8, 15
PRINT#15, "SCRATCH0:filename"
CLOSE 15
```

Now delete EXERCISE 3

```
OPEN 15, 8, 15
PRINT#15, "SCRATCH0:EXERCISE 3"
CLOSE 15
```

Check the contents of the directory.

## Reorganising the disk

When you have saved, copied and deleted files, the disk may need to be reorganised, so that available space is used economically.

Use these commands:

```
OPEN 15, 8, 15
PRINT#15, "VALIDATE"
CLOSE 15
```

## **Disk protection**

You already know that you can only store data on a disk which has the **WRITE-PERMIT** notch uncovered.

To protect the entire contents of a disk, place a **WRITE-PROTECT TAB** over the notch. This prevents any further data being written to the disk.

## **To print a file saved on disk**

The file must first be loaded into memory. Then follow instructions in chapter 7.

## 9

# Cassette recorder and cassettes

You may omit this chapter if you are using only a disk drive and disks.

When you have entered a program into computer memory, and have trialled that program, you may wish to save it on cassette.

## Saving files on cassette

- 1 Turn machine on.
- 2 Insert a blank cassette into the cassette recorder, rewind the tape to the beginning, then advance the leader tape and set the counter to 0.
- 3 Screen displays **READY** and cursor.
- 4 Type **SAVE "filename"** eg **SAVE "EXERCISE 1"**
- 5 Depress **RECORD** and **PLAY** buttons on recorder, as per screen instructions. The screen goes blank as the program is being saved.
- 6 When the **READY** prompt is displayed on the screen, depress the **STOP** button on the recorder.
- 7 Make a note of the finishing counter number.

Keep a record of programs saved on each cassette, with counter starting and finishing numbers. Provided you keep this record, you can save further programs on the cassette by commencing the next **SAVE** just after the counter number which indicated the finish of the previous program.

## Verify file

If you wish to verify that a program has been stored in the correct manner on the cassette:

- 1 Rewind cassette to starting counter number.

- 2** Type `VERIFY`
- 3** If stored correctly, the message `OK READY` will appear on the screen.
- 4** If not stored correctly, the message `VERIFY ERROR READY` will appear. This may indicate that the recorder is faulty, or that the tape is of poor quality, or that the tape heads need cleaning.

## Loading a file from cassette

- 1** Rewind the cassette to the counter starting number.
- 2** Type `NEW` to clear the memory.
- 3** Type `LOAD "filename"` and press `RETURN`.  
Follow the screen instructions. The cursor will disappear.
- 4** The screen goes blank for a few seconds.
- 5** The screen will display `SEARCHING FOR "filename"`.
- 6** After the screen displays `FOUND "filename"`, the screen will again go blank and then display `LOADING`.
- 7** When the program is loaded, the screen will display `READY` and the cursor.
- 8** Stop the cassette recorder.

To stop the searching or loading process, press `RUN/STOP`.

## Running the program

Now that the contents of the tape are safely stored in the computer memory, you could type either:

`LIST` or

`RUN`

## Printing a file saved on cassette

The file must first be loaded into memory. Then follow instructions in chapter 7.



# 10

## Assignment of data to variables

### INPUT READ DATA

Now that you know how to save your programs on disk or cassette, you should save each program, selecting an appropriate filename.

### Assigning data to variables

In previous programs you assigned:

- numeric data to numeric variables

eg LET A% = 24	or	A% = 24
LET B = 76.67	or	B = 76.67

- string data to string variables

eg LET A\$ = "NAME"	or	A\$ = "NAME"
LET C\$ = "COST"	or	C\$ = "COST"

Data can also be assigned by using:

INPUT, READ and DATA

The same rules apply for assigning data by these two methods, ie numeric data must use a numeric variable and string data must use a string variable.

### INPUT statement

The INPUT statement allows data to be entered at the keyboard while the program is running. This means that you, as the operator, can enter data as required.

The RUN of the program is stopped to allow data entry. The RUN will not continue until the operator supplies the requested data. If the operator responds by only pressing RETURN, the variable will remain unchanged.

Enter

```
10 PRINT "ENTER YOUR NAME"
20 INPUT N$
30 INPUT "ENTER TODAY'S DATE DD/MM/YY ";D$
40 PRINT "ENTER A NUMBER 1-10"
50 INPUT N1%
60 INPUT "ENTER A NUMBER 50-100";N2%
70 PRINT "HELLO ";N$: PRINT
80 PRINT "THE DATE IS ";D$: PRINT
90 PRINT "YOUR NUMBERS ARE ";N1%;" AND ";N2%
9999 END
```

The INPUT statement can be used as in:

- Statements 10 and 20 — when the program is RUN, a question mark appears on the screen after the prompt message “Enter your name”. You then enter the requested data. Quotation marks are not required around the input entries.
- Statement 30 — when the program is RUN, the prompt message “Enter today’s date DD/MM/YY” appears on the screen, with a following question mark. You then enter the requested data. Quotation marks are not required around the input entries.

LIST and RUN

**1** Enter this program:

```
10 REM USING INPUT STATEMENT
20 PRINT "ENTER YOUR NAME"
30 INPUT N$
40 INPUT "ENTER TODAY'S DATE DD/MM/YY ";D$
50 PRINT "ENTER TWO PRICES UNDER $10"
60 INPUT P1,P2
70 INPUT "ENTER TWO NUMBERS UNDER 20 ";N1%,N2%
80 PRINT : PRINT
90 PRINT "EXERCISE 8": PRINT
100 PRINT "INVOICE TO:",N$: PRINT
110 PRINT "AS AT",D$: PRINT
120 PRINT "QUANTITY","PRICE","COST": PRINT
130 PRINT N1%,P1,N1% * P1: PRINT
140 PRINT N2%,P2,N2% * P2: PRINT
150 PRINT "TOTAL COST",,N1% * P1 + N2% * P2
9999 END
```

LIST

RUN — Remember that when the prompt message appears on the screen, you must enter the requested data. If more than one entry is requested (eg statements 50 and 70), separate the entries with a comma.

The advantage of a program which uses the INPUT statement is that the program will not need alteration to insert a different name, date or other details. These details are supplied by the operator while the program is running.

## Exercise 9

Refer to exercise 7 in chapter 7. Write a program, using INPUT statements, which could be used to find the area of any rectangle.

## Exercise 10

Using INPUT statements, write a program which will print the square root of any number entered.

## READ and DATA statements

Another way to assign data is to use READ and DATA statements. Two separate statements must be used. The READ statement assigns the variables to the data items, while the DATA statement supplies the data. In the DATA statement, it is not essential for string data to be enclosed in quotation marks.

Enter

```
10 READ A%,B$
20 PRINT A%;B$
30 DATA 250, " DOLLARS"
9999 END
```

LIST and RUN

When this program is run, the system automatically assigns the variables to the data — the first variable (A%) to the first data item (250), the second variable (B\$) to the second data item (DOLLARS).

It is essential that the type of variable (numeric or string) in the READ statement matches the type of data (numeric or string) in the DATA statement.

- 1 Enter this program to assign numeric data (15, 32, 56) to numeric variables (A%, B%, C%) by means of READ and DATA statements.

```
10 REM ASSIGNMENT OF NUMERIC DATA
20 PRINT "EXERCISE 11"
30 PRINT "*****": PRINT
40 READ A%,B%,C%
50 PRINT A%,B%,C%
60 PRINT : PRINT
70 PRINT A%,B%,C%
80 DATA 15, 32, 56
9999 END
```

LIST and RUN

What was the effect of statement 60? (It printed two blank lines.)

In running this program, the computer assigned the value of the first DATA item (15) to the first variable (A%); the second DATA item (32) to the second variable (B%); and the third DATA item (56) to the third variable (C%).

- 2 Now: change exercise 11 to exercise 12  
insert a statement to print your name  
change statement 60 so that one blank line will be printed  
insert a statement to instruct that  $D\% = A\% + B\% + C\%$   
insert a statement to print that " $A\% + B\% + C\% =$ ";D%

LIST and RUN

- 3 Refer to the program for exercise 4 in chapter 7.

Work out how to change this program so that the values of N% and P are assigned by READ and DATA statements.

Enter your program as exercise 13.

**Remember:** NEW LIST RUN

- 4 Refer to exercise 7 in chapter 7.

Write a program to solve that problem, using READ and DATA statements to assign the variables for length and width.

Enter your program as exercise 14.

**Remember:** NEW LIST RUN

- 5 In the following exercise, string data is assigned to string variables, using the READ and DATA statements.

Enter this program.

```
10 REM ASSIGNMENT OF STRING DATA
20 PRINT "EXERCISE 15": PRINT
30 READ H$,N$,D$
40 READ A$,J$,P$,S$
50 READ W$,X$,U$,Z$
60 PRINT TAB( 6);H$: PRINT : PRINT
70 PRINT N$,D$: PRINT
80 PRINT A$,W$: PRINT J$,X$
90 PRINT P$,Y$: PRINT S$,Z$
100 PRINT : PRINT
110 PRINT TAB( 6);H$: PRINT
120 PRINT D$,N$: PRINT : PRINT
130 PRINT X$,J$: PRINT Z$,S$
140 PRINT W$,A$: PRINT Y$,P$
150 DATA "BIRTHDAY LIST","NAME","DATE"
160 DATA "ALAN","JUDITH","CARLO","ANNA"
170 DATA "17 JUL","30 JAN"
180 DATA "12 OCT","14 APR"
9999 END
```

LIST and RUN

**Remember:** The first string data item (BIRTHDAY LIST) is assigned to the first string variable (H\$), the second string data item (NAME) to the second string variable (N\$), and so on.

Once the data has been assigned to variables, those variables may be used in any order and — if required — more than once. Notice that your program used the same data to produce two lists, one in alphabetical order by name, the other in chronological (date) order.

- 6 Write a program which will print a list of the titles of three songs, and the names of the singers for each song. Use **READ** and **DATA** statements to assign the variables. Provide a heading and column headings.

Enter your program as exercise 16.

- 7 Write a program (exercise 17) which will print the following table:

#### CAPITAL CITIES

STATE	CAPITAL
QUEENSLAND	BRISBANE
VICTORIA	MELBOURNE
WESTERN AUSTRALIA	PERTH

Use **READ** and **DATA** statements to assign the variables, other than the heading and column headings.

**Remember:** NEW LIST RUN

- 8 Write a program which will print the names of four suburbs (string data) and the postcodes for those suburbs (numeric data). Use **READ** and **DATA** statements to assign variables. Include a heading and column headings for the output.

Enter your program as exercise 18.

- 9 Write a program to produce the following output. Use **READ** and **DATA** statements to assign variables, other than headings.

#### STUDENT RESULTS

NAME	SUBJECT	PER CENT
ROSIE	ENGLISH	76
GIORGIO	SCIENCE	81
ANGELA	ENGLISH	69
WILLIAM	MATHS	73

Enter your program as exercise 19.

# 11

## Loops

### GOTO IF... THEN FOR and NEXT

When the solution to a problem requires that some instructions should be repeated, a **loop** may be used. This simply means that when the program run reaches a particular statement, it will automatically return to a previous statement and repeat the instructions given.

### GOTO statement

The GOTO statement instructs the computer to **go to** another statement number. In the following program, statement 70 instructs the computer to go back to statement 50 and repeat the instructions. This is called a **loop**.

The GOTO statement is called an **unconditional** statement because the computer must obey.

1 Enter this program:

NEW

```
10 REM CALCULATING THE PRODUCT OF TWO VALUES
20 REM N=VALUE 1, P = VALUE 2
30 PRINT "EXERCISE 20": PRINT
40 PRINT "VALUE 1", "VALUE 2", "PRODUCT": PRINT
50 READ N%, P%
60 PRINT N%, P%, N% * P%
70 GOTO 50
80 DATA 10, 20, 15, 7, 23, 6, 4, 81
9999 END
```

LIST and RUN

After the program has run, notice the error message which indicates

OUT OF DATA. The computer followed your instructions until all the data (statement 80) had been used. Then — following your instruction in statement 70 — the computer went back to statement 50, but could find no more data to read. The error message explains to you why the computer could not follow your instructions any further.

- 2 Design a short program, using the GOTO statement, to find the price of each article.

14 articles	total cost \$310.66
27 articles	total cost \$866.97
21 articles	total cost \$176.82

Call your program exercise 21. Select your own column headings. Price will equal cost divided by number.

**Remember:** NEW (unless you can adapt exercise 20)  
LIST and RUN

- 3 Here is another example of a GOTO statement. Remember: GOTO means that the computer must obey the command — it is an unconditional statement.

In this program, the GOTO statement will cause a never-ending loop, so it will be up to you to stop the run execution when you have seen how it works.

To stop the run execution, use RUN/STOP

Enter this program:

NEW

```
10 REM GO TO AND CONTINUOUS LOOP
20 PRINT "EXERCISE 22": PRINT
30 PRINT "YOUR NAME": PRINT
40 READ L%
50 PRINT L%
60 L% = L% + 2
70 GOTO 50
80 DATA 2
9999 END
```

Look at this program and try to work out why the GOTO statement will cause a never ending loop.

LIST and RUN

**Remember:** ● stop the run with RUN/STOP  
● resume the run by typing CONT

### Notes:

- a When you stopped the program run, you received a message BREAK IN (no) . This means that when you depressed RUN/STOP the run of the program was stopped at statement (no).

- b** What does statement 60 mean?  
 $L = L + 2$  is actually an instruction to the computer: 'Let the value of  $L$  be replaced by the value of  $L + 2$ .'
- c** If you want to run the program again,
- type `RUN` to commence the program run again
  - type `CONT` to continue the program run
- but remember to use `RUN/STOP` to stop the run.
- d** To slow down the run, hold `CTRL` key — when `CTRL` is released the program resumes normal speed.

## IF ... THEN statement

The `IF ... THEN` statement is a **conditional** statement, because it instructs the computer:

`IF` (a certain condition is met) `THEN` (go to another statement).

Remember the continuous loop in exercise 22? Now you are going to insert a condition that will end that loop.

- 1** Enter this program:

`NEW`

```

10  REM  IF...THEN STATEMENT
20  PRINT "EXERCISE 23": PRINT
30  PRINT "YOUR NAME": PRINT
40  READ L%
50  PRINT L%
60  L% = L% + 2
70  IF L% > 20 THEN 9999
80  GOTO 50
90  DATA 2
9999 END

```

The sign `>` means 'greater than', so statement 70 instructs the computer:

`IF L` is greater than 20, `THEN` go to statement 9999.

`LIST` and `RUN`

- 2** Amend exercise 23 by:

- changing in statement 50 to `PRINT L,` (what effect will the comma have?)



- Changing statement 60 to  $L = L - 2$
- Changing statement 70 to IF  $L < 0$  THEN 9999
- Changing statement 90 to DATA 100

The sign  $<$  means 'less than' so statement 70 will now instruct:

IF  $L$  is less than 0, THEN go to statement 9999.

LIST and RUN

- 3** Look carefully at this program and try to understand it before you enter it.

NEW

```

10 REM IF...THEN STATEMENT
20 REM P = PRICE, R = RATE
30 PRINT "EXERCISE 24": PRINT
40 PRINT "YOUR NAME": PRINT
50 PRINT TAB( 10); "SALES TAX SCHEDULE": PRINT
60 PRINT "PRICE", "RATE %", "TAX": PRINT
70 READ P%, R
80 PRINT P%, R, P% * R
90 P% = P% + 5
100 IF P% > 105 THEN 9999
110 GOTO 80
120 DATA 20, .10
9999 END

```

Have you worked out how this program will run?

Enter the program.

LIST and RUN

- 4** Design a short program (exercise 25) to produce the following output.  
Use READ and DATA, GOTO, IF ... THEN statements.

#### SCHEDULE OF CASUAL RATES

HOURS WORKED	HOURLY RATE	WAGES DUE (HOURS * RATE)
2	6.80	
4	6.80	
6	6.80	
8	6.80	
up to 40		

- 5 Enter this program, which will allow you to INPUT two numbers, PRINT both numbers, and determine which is the larger number.

NEW

```
10 REM DETERMINING LARGER NUMBER
20 REM N1% = NUMBER, N2% = NUMBER
30 PRINT "EXERCISE 26": PRINT
40 PRINT "YOUR NAME": PRINT
50 INPUT "ENTER TWO INTEGER NUMBERS ";N1%,N2%: PRINT
60 PRINT "NUMBERS ARE: ",N1%,N2%: PRINT
70 IF N1% > N2% THEN 100
80 PRINT "LARGER NUMBER IS:",N2%
90 GOTO 9999
100 PRINT "LARGER NUMBER IS:",N1%
9999 END
```

LIST and RUN

- 6 Write a program (exercise 27) using INPUT, IF . . . THEN and GOTO statements, which will:

- INPUT a student's name
- INPUT exam marks for that student for three exams where the maximum marks for each exam are 25, 35 and 40.
- calculate the total marks
- print student name, total marks, and  
PASS (if total marks are 50 or above)  
FAIL (if total marks are less than 50)

Supply a suitable heading and column headings for your output.

## FOR and NEXT statements

The number of times an instruction is performed can be controlled by FOR and NEXT statements, which cause the program to loop.

In the following example, the instructions between statements 60 and 90 will be performed ten times.

FOR sets the counter (C) to start at a selected number (eg 1) and to finish at a selected number (eg 10).

STEP sets the selected number by which the counter (C) will be increased each time the instruction is performed (eg STEP 1 means increase the value of the counter by + 1).

NEXT increases the counter (C) by the step shown (+ 1). If the counter is less than the selected finishing number (eg 10), the statements in the loop are repeated. If the counter is equal to the selected finishing number (eg 10), the loop is completed. The program would then continue from the statement following the NEXT statement.

**1 Study this program.**

NEW

```
10 REM   FOR AND NEXT STATEMENTS
20 REM   C = COUNTER FOR LOOP
30 PRINT "EXERCISE 28": PRINT
40 PRINT "YOUR NAME": PRINT
50 READ A%,B%
60 FOR C = 1 TO 10 STEP 1
70 PRINT A%,B%
80 A% = A% + B%
90 NEXT C
100 DATA 10, 2
9999 END
```

Statement 60 (FOR C = 1 TO 10 STEP 1) is the starting point for the loop. The counter C will start at 1.

Statement 90 (NEXT C) is the instruction to go to statement 60 and increase the counter by the STEP shown (in this program STEP 1). So the counter becomes 2.

The program loops (works) from statement 60 to statement 90 until the counter = 10. The loop will then end.

Enter, LIST and RUN this program.

**2 Now make these changes:**

Change to exercise 29

Change the FOR statement to STEP 2.

Change  $A\% = A\% + B\%$  to  $A\% = A\% * B\%$

LIST and RUN

**3 It is also possible to have the loop counter decrease in value by using a minus step. In this case, the selected starting point for the loop must be larger than the selected finishing number.**

eg FOR C = 100 TO 20 STEP -5

The counter (C) will start at 100 and decrease by 5 each time the loop is executed. When the counter is equal to 20, the loop is stopped.

NEW

```
10 REM   USING STEP MINUS
20 PRINT "EXERCISE 30": PRINT
30 FOR C = 100 TO 20 STEP - 5
40 PRINT C
50 NEXT C
60 PRINT
70 PRINT "EXAMPLE OF MINUS STEP"
9999 END
```

LIST and RUN

Notice that in this example the actual decreasing value of the counter has been printed.

- 4** What do you think will happen when you RUN this program?

NEW

```
10 REM SIXES AND SEVENS
20 REM C = COUNTER
30 PRINT "EXERCISE 31": PRINT
40 PRINT "YOUR NAME": PRINT
50 READ A%,B%
60 FOR C = 200 TO 0 STEP - 1
70 PRINT A%;B%;
80 NEXT C
90 DATA 6, 7
9999 END
```

Enter this program.

LIST and RUN

- 5** In this example, the READ and PRINT statements are contained within the loop. The first time through the loop, the first two data items are read. The second time through the loop, the third and fourth data items are read, and so on.

NEW

```
10 REM READ INSIDE LOOP
20 REM C = COUNTER
30 REM S$=STATE, C$=CAPITAL
40 PRINT "EXERCISE 32": PRINT
50 PRINT "CAPITAL CITIES": PRINT
60 PRINT "STATE","CITY": PRINT
70 FOR C = 1 TO 3 STEP 1
80 READ S$,C$
90 PRINT S$,C$
100 NEXT C
110 DATA "TASMANIA","HOBART"
120 DATA "S.AUST.","ADELAIDE"
130 DATA "N.S.W.","SYDNEY"
9999 END
```

LIST and RUN

Did you notice that this program produces similar output to exercise 17 in chapter 10? By using the loop, only two variables were used, and only one PRINT statement.

- 6** Write a program (exercise 33) using READ and DATA statements, and a FOR and NEXT loop, to produce the same output as exercise 18 in chapter 10. Use only two variables, and contain the READ and PRINT statements within the loop.
- 7** Refer to the output for exercise 19 in chapter 10. Use READ and DATA, and FOR and NEXT statements, to produce that output. Use

three variables only and contain the **READ** and **PRINT** statements inside the loop.

Enter your program as exercise 34.

- 8** Write a program which will print a list of the names, suburbs and phone numbers of six people.

Enter your program as exercise 35.

## Variables — real or integer?

You already know that a numeric integer variable (eg **N%**) represents a whole number and that a numeric real variable (eg **N**) represents a number which may have a decimal portion. In many of the programs you will write, you will either be using a real number or the results of your calculations will be a real number. For this reason, it may be preferable for you to use numeric real variables to represent all numbers, both integer and real. For convenience, the programs in the remainder of this book will use numeric real variables.

# 12

## Loops — multiple and nested

A program may use more than one loop to repeat instructions. The loops may be separate parts of the program, or they may be nested (ie one inside another).

### Multiple loops

#### 1 Enter NEW

```
10  FOR C = 1 TO 3 STEP 1
20  PRINT C
30  NEXT C
40  PRINT "END OF FIRST LOOP": PRINT
50  FOR K = 2 TO 10 STEP 2
60  PRINT K
70  NEXT K
80  PRINT "END OF SECOND LOOP"
9999  END
```

LIST and RUN

- 2** A transport company needs to calculate the number of trips to move five quantities of soil (1890 kg per trip) at the cost of 35c per kg. Also required is a separate table showing earnings for six employees, based on an hourly rate of \$11.70. This program (exercise 36) uses two loops to solve their problem. The data statements contain the five quantities of soil and the hours worked by six employees.

NEW

```
10 REM USING TWO LOOPS
20 REM Q = QUANTITY, C = COST, H = HOURS
30 PRINT "EXERCISE 36": PRINT
40 PRINT "QUANTITY","TRIPS","COST": PRINT
50 FOR C = 1 TO 5 STEP 1
60 READ Q
70 PRINT Q,Q / 1890,Q * .35
80 NEXT C
90 PRINT
100 PRINT "HOURS","EARNINGS": PRINT
110 FOR K = 1 TO 6 STEP 1
120 READ H
130 PRINT H,H * 11.70
140 NEXT K
150 DATA 13230, 21735, 4347, 11340, 18711
160 DATA 27, 14, 31, 8, 40, 21
9999 END
```

LIST and RUN

- 3** Write a program (exercise 37) which will
- a** convert five Celsius temperatures to Fahrenheit temperatures  
 $F = (C/5) * 9 + 32$
  - b** convert five Fahrenheit temperatures to Celsius temperatures  
 $C = (F - 32) * 5/9$

Use two loops, and print two separate tables with suitable headings.

## Nested loops

- 1** The term **nested loops** describes two loops, one of which is totally enclosed within the other loop.

NEW

```
10 FOR C = 1 TO 3 STEP 1
20 PRINT "LOOP ";C: PRINT
30 FOR K = 1 TO 4 STEP 1
40 PRINT K
50 NEXT K
60 PRINT
70 NEXT C
9999 END
```

LIST and RUN

Notice that the FOR and NEXT statements for counter K (statements 30 and 50) are totally enclosed within the FOR and NEXT statements for counter C (statements 10 and 70).

- 2** In this program, a payroll is being calculated for four employees, based on daily hours worked over a period of five days. The hourly rate is \$8.75. If the employees work more than 35 hours, they are paid at time and a half.

Using nested loops, the first loop reads and prints the employee name. The second loop reads the hours, adds the hours, calculates and prints the earnings.

Notice that total hours (T) is set to zero each time a name is read.

Enter, LIST and RUN this program.

NEW

```
10 REM CALCULATING PAYROLL
20 REM N$=NAME, H=HOURS, T=TOTAL HOURS
30 PRINT "EXERCISE 38": PRINT
40 PRINT TAB( 16); "PAYROLL": PRINT
50 PRINT "NAME", "HOURS", "EARNINGS": PRINT
60 FOR C = 1 TO 4 STEP 1
70 T = 0
80 READ N$
90 PRINT N$
100 FOR K = 1 TO 5 STEP 1
110 READ H
120 T = T + H
130 NEXT K
140 IF T > 35 THEN 170
150 PRINT T, T * 8.75
160 GOTO 180
170 PRINT T, (35 * 8.75) + ((T - 35) * (8.75 * 1.5))
180 PRINT
190 NEXT C
200 DATA "MARINO", 7, 6.5, 8, 6, 4
210 DATA "HUNTER", 7, 8, 6, 7, 7
220 DATA "POPOVIC", 8, 7.5, 8, 8.5, 6
230 DATA "RILEY", 4, 6, 5, 9, 4
9999 END
```

- 3** Refer to exercise 27 in chapter 11. Write a program (exercise 39) which will READ the data for five students (name and three results), and print a table of name, total marks, and pass or fail. You should use two loops. Provide suitable heading and column headings.



4 Write a program (exercise 40) which will print a table showing the credit terms offered by a store, on each of five different prices. Your output should be:

CREDIT TERMS			
PRICE	DEPOSIT	SIX	TWELVE
\$	10%	MONTHS	MONTHS
(P)	(D)		
	$(P \cdot .10)$	$(P - D)/6$	$(P - D)/12$

# 13

## How to prepare a program

So far, the programs you have used have all been very simple. If you plan to develop your own programs, you will need to learn how to solve the problem and then transfer this solution to your program.

To solve any problem, you should first divide it into smaller sections to make the solution easier.

### Steps to follow

- 1** Consider the problem carefully.
- 2** Select the format you require for your answer.
- 3** Identify the data you have been given.
- 4** Assign variables as required.
- 5** Decide what calculations will be needed.
- 6** Decide what decisions need to be made.
- 7** Prepare guidelines.
- 8** Write the program.

Now, put these steps into action with a simple exercise.

The question for exercise 3 in chapter 7 could have stated:  
Write a program that will print:

### EXERCISE 3

\* \* \* \* \*

Your name

\* \* \* \* \*

$$A\% = 50$$

$$B\% = 10$$

$$C\% = 12$$

$$A\% + B\% * C\% = (A\% + B\% * C\%)$$

$$A\%/B\% * C\% = (A\%/B\% * C\%)$$

#### Step 1

Consider the problem

Read the question carefully

#### Step 2

Select a layout

Already available

#### Step 3

Identify the given data

50

10

12

#### Step 4

Assign variables

A% B% C%

#### Step 5

What calculations?

$A\% + B\% * C\%$

$A\%/B\% * C\%$

#### Step 6

What decisions?

None in this example

#### Step 7

Prepare guidelines

Use a flowchart or list the guidelines as a series of short sentences

#### Step 8

Write the program from your guidelines

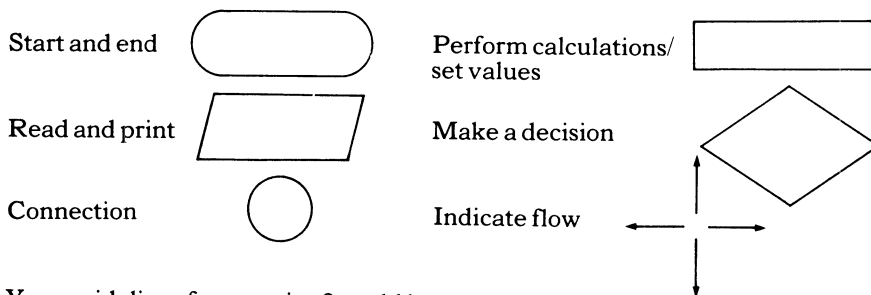
# Guidelines

In preparing guidelines, you may use either:

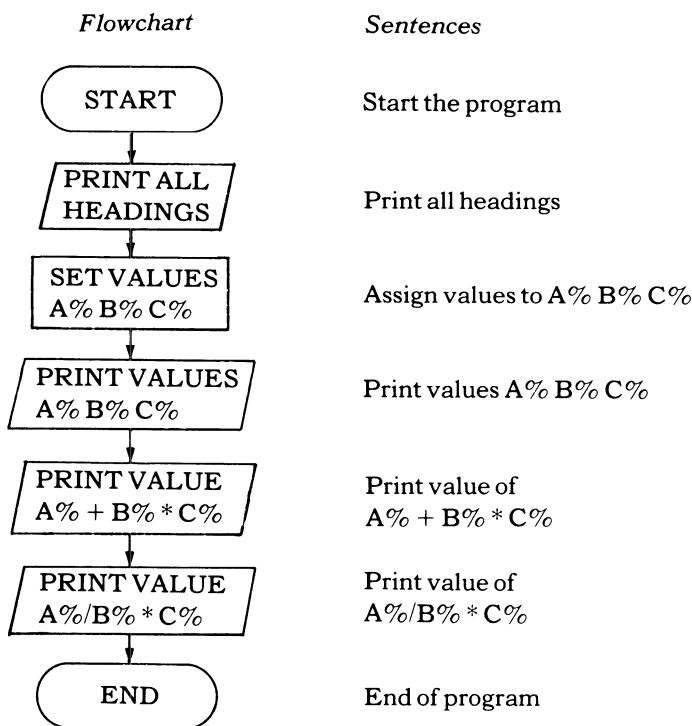
- a flowchart — a diagram of the logical steps needed, or
- short sentences — to outline each step.

For the exercises in this chapter, use both methods and then decide which approach you prefer.

When using a flowchart, the following shapes are often used to represent different steps.



Your guidelines for exercise 3 could be:



**Note:** REM statements are not shown in either the flowchart or the sentences. REMark statements are used to identify and clarify your program.

1 By following the examples, you should now be able to write guidelines for exercise 4 from this question.

Write a program which will print:

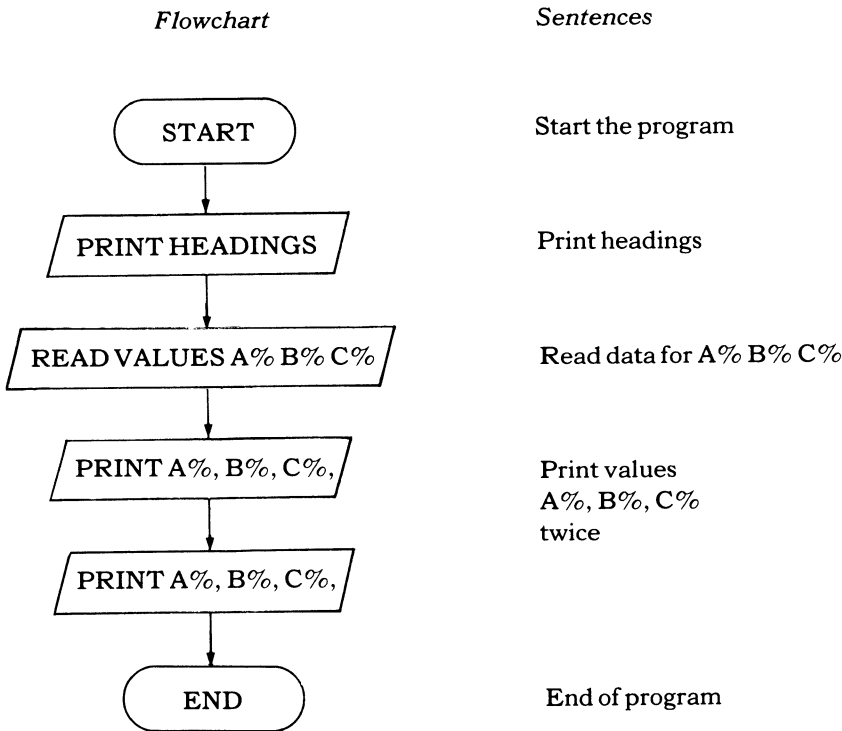
EXERCISE 4  
\* \* \* \* \*

Your name  
\*\*\*\* \*\* \*\*

NUMBER	PRICE	COST
48	17.58	(Number * Price)

2 Now prepare guidelines for exercises 5, 6 and 7 from chapter 7.

3 Here are the guidelines for exercise 11 from chapter 10.



**Note:** DATA statements are not shown in the flowchart.

4 Prepare guidelines for exercise 13 from chapter 10.

5 This question refers to exercise 23 from chapter 11.

Write a program that will print:

### EXERCISE 23

YOUR NAME

2

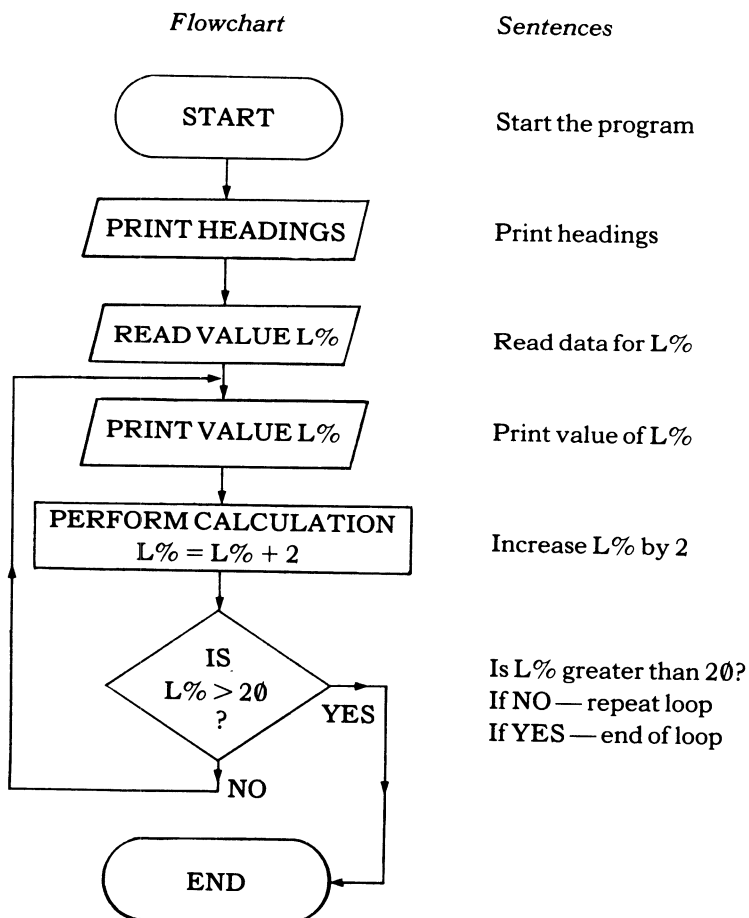
4

6

up to

20

Here are the guidelines:



**Note:** This program will loop until the value of L% is greater than 20.

6 Prepare guidelines for exercises 24 and 25 from chapter 11.

7 The question for exercise 28 from chapter 11 could state:  
Write a program that will print:

**EXERCISE 28**

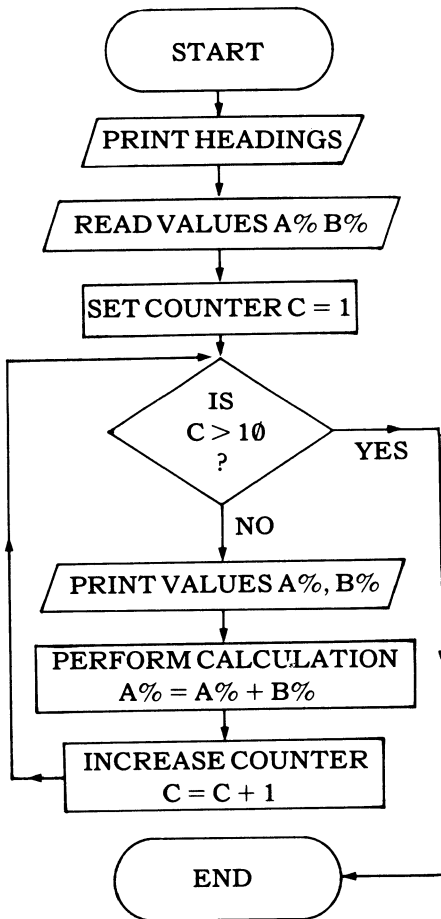
Your name

10	2
12	2
14	2
16	2

up to	
28	2

Here are the guidelines:

*Flowchart*



*Sentences*

Start the program

Print headings

Read data for A% and B%

Set the counter initially to 1 so that the number of loops may be counted.

Is counter greater than 10?

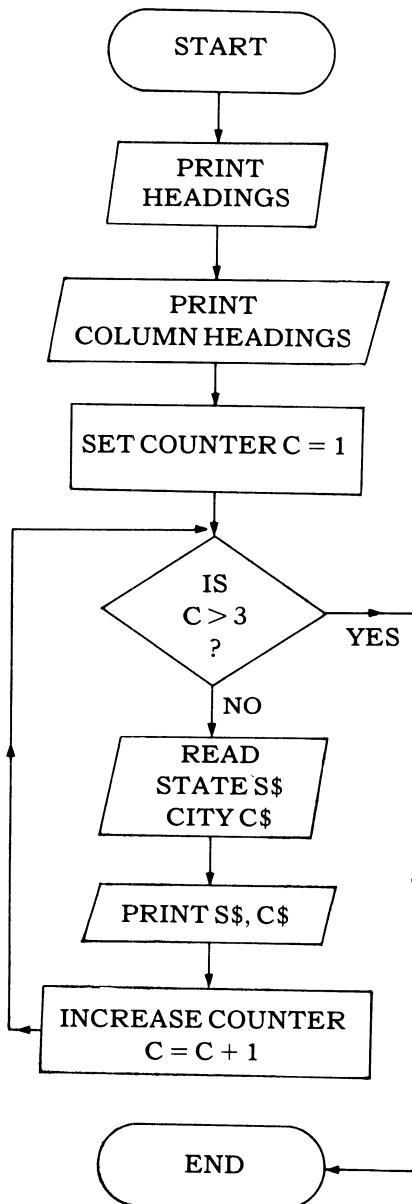
If NO — print A%, B%  
increase A% by B%  
increase counter by 1  
repeat loop

If YES — end of loop

**Note:** The program will loop until the value of C is greater than 10.

8 Here are the guidelines for exercise 32 from chapter 11.

*Flowchart*



*Sentences*

Start the program

Print headings

Print column headings

Set the counter initially to 1 so that the number of loops may be counted.

Is the counter greater than 3?

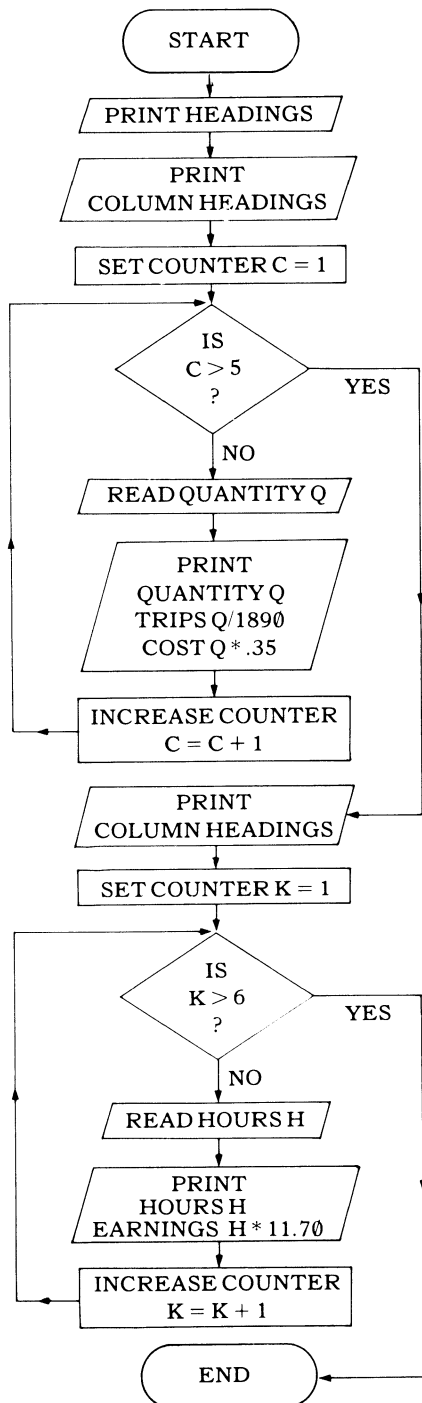
If NO – read S\$, C\$  
print S\$, C\$  
increase counter by 1  
repeat loop

If YES — end of loop



9 These guidelines apply to exercise 36 from chapter 12.

### Flowchart



### Sentences

Start the program

Print headings

Print column headings

Set the counter to 1

Is the counter greater than 5?

If NO — read Quantity Q  
print Quantity Q  
print Trips  $Q/1890$   
print Cost  $Q \cdot .35$   
increase counter by 1  
repeat loop

If YES — next section of program

Print column headings

Set the counter to 1

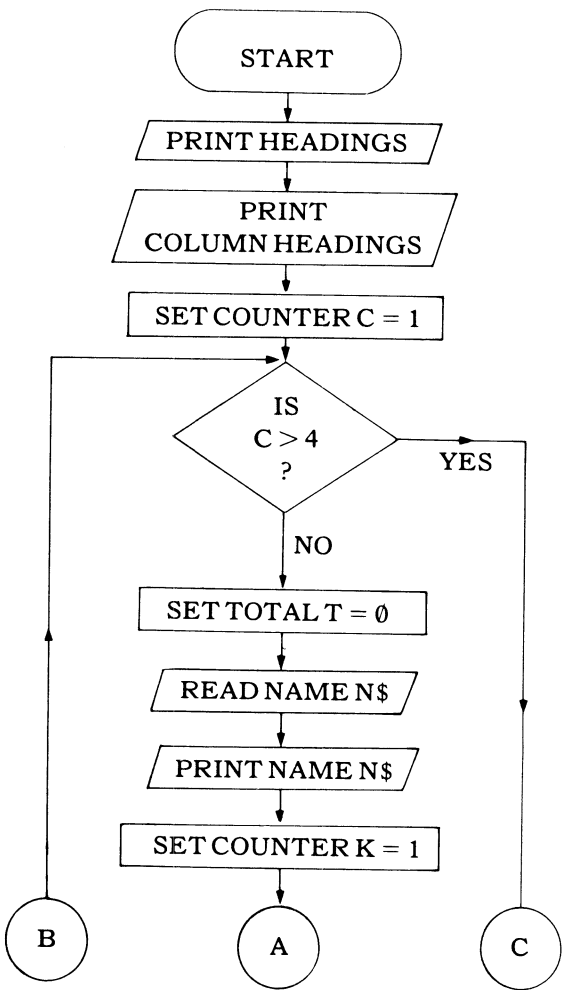
Is the counter greater than 6?

If NO — read Hours H  
print Hours H  
print earnings  $H \cdot 11.70$   
increase counter by 1  
repeat loop

If YES — end of loop

10 Exercise 38 from chapter 12 could have the following guidelines.

*Flowchart*



*Sentences*

Start the program

Print headings

Print column headings

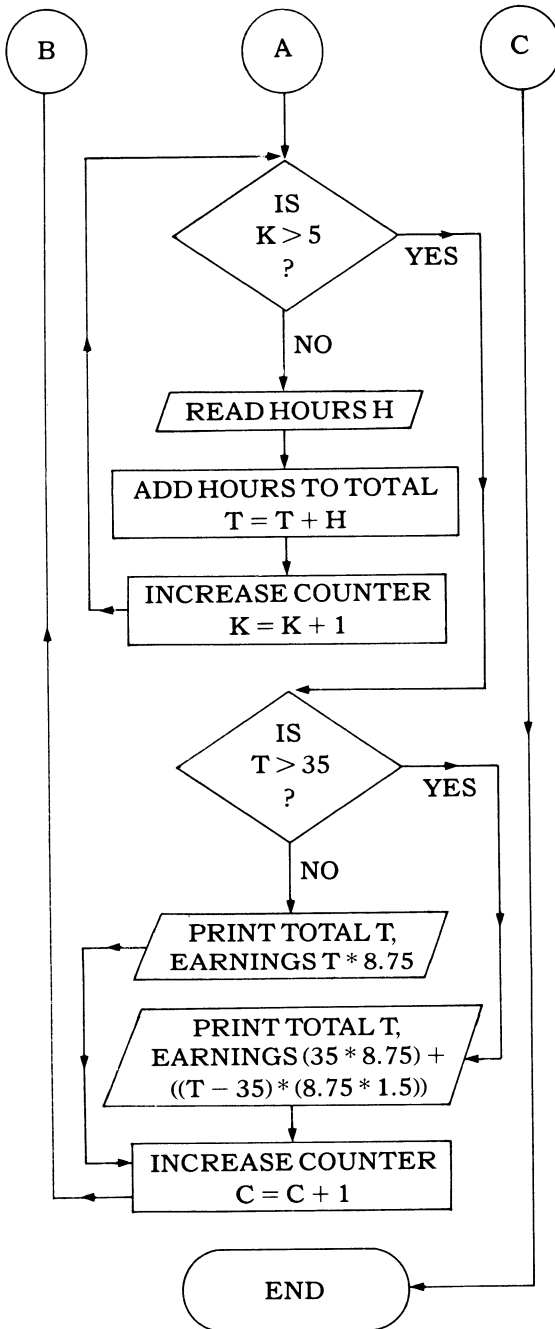
Set the counter C to 1

Is the counter (C)  
greater than 4?

If NO — set Total T to 0  
read Name N\$  
print Name N\$  
set counter K to 0  
Is counter (K) > 5?

If NO — read Hours  
add Hours H  
to Total T  
increase K by 1  
repeat K loop

cont.



If YES — is total > 35?

If NO— print total  
print earnings  
increase C by 1  
repeat C loop

If YES — print total  
print earnings  
plus overtime  
increase C by 1  
repeat C loop

If YES — end of loop

## 11 Exercise 41

You have purchased a car for \$8750. You paid a deposit of \$2030 and obtained a loan for the balance over a term of 24 months. You have agreed to make a monthly payment of \$280.00.

Use the following steps to solve the problem:

- Prepare guidelines
- Write the program
- Enter the program
- RUN the program

Your answer should show all details, and should include a table showing month (1 to 24), payment and balance (after payment).

Some hints to help:

- a Use a counter to indicate the months 1 to 24.
- b The value of the counter can be printed.
- c First balance will equal (cost minus deposit).
- d Each following balance will equal (balance minus payment).

Good programming!

**Note:** Programs for any question may vary in the order of steps. The only way to test whether a program is written correctly is to run the program.

If an accurate result is obtained, in the correct format, then the program is correct.

# 14

## More statements

This chapter introduces a number of additional statements and the work involved is slightly more difficult.

Work the following examples to gain an understanding of the statements. LIST and RUN each program.

### RESTORE statement

The RESTORE statement instructs the computer to read data again, from the first data statement.

```
10 READ A,B
20 PRINT A,B
30 READ A,B
40 PRINT A,B
50 RESTORE
60 READ A,B
70 PRINT A,B
80 DATA 67, 102, 48, 76
9999 END
```

(Try this without the RESTORE statement!)

### Subroutine

The instruction GOSUB directs the computer to go to the **subroutine** section of the program, which is identified by the statement number following GOSUB. The subroutine section is then executed and the final statement in the subroutine is RETURN, which directs the computer to return to the statement following the GOSUB statement.

```

10  GOSUB 50          (GO to the start of SUBroutine 50)
20  PRINT 2,4,6
30  PRINT
40  GOTO 9999
50  PRINT 1,3,5      (start of subroutine)
60  RETURN           (RETURN to the statement after GOSUB)
9999  END

```

## Dummy data

Dummy data is data which is included to signify the end of data. This is useful when the amount of data is subject to change.

```

10  READ A
20  IF A = - 111 THEN 9999
30  PRINT A,
40  GOTO 10
50  DATA 15, 21, 36, 18, 12, -111
9999  END

```

## LEN (length) statement

The LEN statement determines the number of characters in a variable.

```

10  A$ = "AUSTRALIA"
20  B$ = "COMMODORE MICROCOMPUTER"
30  A = LEN (A$):B = LEN (B$)
40  PRINT A$,A
50  PRINT B$,B
9999  END

```

## ON...GOTO statement

The ON...GOTO statement allows a program to branch to one of a number of other statements, depending on the input.

```

10 INPUT "GUESS A NUMBER BETWEEN 1 AND 5 ";A
20 IF A < 1 OR A > 5 THEN 90
30 ON A GOTO 40,50,60,70,80
40 PRINT "YOU GUESSED NUMBER 1": GOTO 100
50 PRINT "YOU GUESSED NUMBER 2": GOTO 100
60 PRINT "YOU GUESSED NUMBER 3": GOTO 100
70 PRINT "YOU GUESSED NUMBER 4": GOTO 100
80 PRINT "YOU GUESSED NUMBER 5": GOTO 100
90 PRINT : PRINT "WRONG NUMBER - TRY AGAIN": GOTO 10
100 PRINT : PRINT "DO YOU WANT TO GUESS AGAIN?"
110 INPUT "ENTER Y OR N ";G$: PRINT
120 IF G$ = "Y" THEN 10
9999 END

```

## ON...GOSUB statement

This is very similar to the ON...GOTO statement in that the program will branch to one of a number of subroutines depending on the input. The variable used for the input must be a numeric variable.

```

10 PRINT "1 READ      2 INPUT": PRINT
20 PRINT "3 PRINT     4 SAVE": PRINT
30 INPUT "SELECT BY NUMBER ";N: PRINT
40 IF N < 1 OR N > 4 THEN 100
50 ON N GOSUB 60,70,80,90
60 PRINT "SUBROUTINE TO READ": GOTO 9999
70 PRINT "SUBROUTINE TO INPUT": GOTO 9999
80 PRINT "SUBROUTINE TO PRINT": GOTO 9999
90 PRINT "SUBROUTINE TO SAVE": GOTO 9999
100 PRINT "INCORRECT NUMBER": PRINT : GOTO 10
9999 END

```

## INT (integer) statement

The INT statement will return a whole number, less than or equal to a decimal number.

```

10 A = 17.6:B = 21.1
20 C = - 2.6:D = - 9.1
30 PRINT A, INT (A)
40 PRINT B, INT (B)
50 PRINT C, INT (C)
60 PRINT D, INT (D)
9999 END

```

## LEFT\$, MID\$, RIGHT\$ statements

These statements will allow you to extract selected characters from a variable.

```
10  A$ = "AUSTRALIA"
20  PRINT LEFT$ (A$, 4)    (first 4 characters)
30  PRINT RIGHT$ (A$, 3)  (last 3 characters)
40  PRINT MID$ (A$, 5, 2)  (starts at character 5; prints 2 characters)
50  PRINT MID$ (A$, 6)    (starts at character 6, prints to end)
60  B$ = "25/12/198-"
70  PRINT LEFT$ (B$, 2), RIGHT$ (B$, 4)
80  PRINT MID$ (B$, 4, 2), MID$ (B$, 5)
9999  END
```

## VAL (value) statement

The VAL statement converts a string variable (or part of a string variable) to a numeric value.

```
10  N$ = "1234"
20  N = VAL (N$)
30  PRINT N$, N
40  D$ = "25/12/1983"
50  D = VAL (D$)
60  PRINT D$, D
70  Y = VAL ( RIGHT$ (D$, 2))
80  PRINT D$, Y
90  M = VAL ( MID$ (D$, 4, 2))
100 PRINT D$, M
110 Z = VAL ( LEFT$ (D$, 2))
120 PRINT D$, Z
9999  END
```

## Writing your own program

Now you will progressively develop a program using, as an example, data relating to students.

Work carefully through each section, following the instructions given.

The problem you have been given is to write a program that will return student information in different formats.

### Notes:

- a** Be sure to enter statement numbers exactly as shown, to allow for further statements to be entered as the program is developed.
- b** Keep a printed copy of the program LIST and RUN for comparisons as the program is developed.



- c** When the program is run, the screen display will wrap each line to the next line. However, when the program is displayed by the printer, the output will be in the correct format.

```
10 REM TO PROCESS STUDENT RESULTS
20 REM VARIABLES DESCRIPTION
30 REM DATE D$
40 REM NAME N$
50 REM ADDRESS A$
60 REM PHONE P$
70 REM BIRTHDATE B$
110 PRINT "ENTER CURRENT DATE DD/MM/YY"
120 PRINT : INPUT D$
160 H$ = "PERSONAL DETAILS"
180 PRINT H$
185 PRINT : PRINT D$: PRINT
190 PRINT "NAME","PHONE","BIRTHDATE","ADDRESS": PRINT
200 FOR R = 1 TO 3 STEP 1
210 READ N$,A$,P$,B$
230 PRINT N$,P$,B$,A$
240 NEXT R
245 PRINT
810 DATA "STUDENT 1","4 HILL ROAD MORNINGSIDE"
815 DATA "396.1234","17/10/67"
830 DATA "STUDENT 2","138 MAY ROAD CARINA"
835 DATA "391.7222","26/08/65"
850 DATA "STUDENT 3","21 BAKER STREET CAMP HILL"
855 DATA "399.6214","12/10/66"
9999 END
```

**SAVE** as **STUDENT DETAILS**

**LIST** and **RUN**

**PRINT** the **LIST** and **RUN**

## Adding more data

It would be useful if your program could also provide a separate list of the class and the subjects in which each student is enrolled.

Enter these statements to describe the variables:

```
75 REM CLASS C$
80 REM ENGLISH E$
90 REM MATHS M$
100 REM SCIENCE S$
```

Enter these statements to include the data, which consists of the above variables in the order **C\$, E\$, M\$, S\$**.

(**E** = **Enrolled**, **N** = **Not enrolled**)

```

820 DATA "9B","E","E","N"
840 DATA "9A","E","E","N"
860 DATA "9C","E","N","E"

```

Change statement 210 to READ the extra data.

```
210 READ N$,A$,P$,B$,C$,E$,M$,S$
```

SAVE as STUDENT DETAILS

LIST and RUN

Print the LIST and RUN

Notice that although the extra data has been read, it has not affected the results, as no instruction was given to PRINT that extra data.

## Using the RESTORE statement

In order to print another list containing students' names and subjects, the data must be read again. This is done by using the RESTORE statement, which instructs the computer to read data again, from the first data statement.

Enter these statements:

```

280 H$ = "STUDENT ENROLMENT"
300 PRINT H$
305 PRINT : PRINT D$: PRINT
310 PRINT "NAME","CLASS","ENGLISH","MATHS","SCIENCE"
315 PRINT
320 RESTORE
330 FOR R = 1 TO 3 STEP 1
340 READ N$,A$,P$,B$,C$,E$,M$,S$
360 PRINT N$,C$,E$,M$,S$
370 NEXT R
375 PRINT
590 GOTO 9999

```

SAVE as STUDENT DETAILS

LIST and RUN

Print the LIST and RUN

Your printed result should now show two separate lists, one for personal details and one for student enrolment.

## Using a subroutine

Where the same directions are to be used more than once in a program, it is a good idea to use a subroutine. This will reduce the amount of typing in a long program and ensure that the directions are performed exactly in the same way every time.

Already, in your program, two statements are identical. They are the READ statements 210 and 340.

Enter these statements:

```
700 REM SUBROUTINE TO READ DATA
710 READ N$,A$,P$,B$,C$,E$,M$,S$
720 RETURN
```

Change statements 210 and 340 as shown:

```
210 GOSUB 700
340 GOSUB 700
```

**SAVE as STUDENT DETAILS**

**LIST and RUN**

Print the **LIST** and **RUN**

When the program reaches the first **GOSUB 700** statement (statement 210), it is redirected to statement 700, then to statement 710 to read the data, then to statement 720 which will **RETURN** the program to the statement immediately after the **GOSUB** statement. The program will then continue until the next **GOSUB 700** statement and the process is repeated.

## Using dummy data

Your program has only been designed to process the results for three students. What happens if you are asked to include details for more students? You could of course increase the counter, but you would then need to change every statement containing a counter, each time you had to include extra data.

Here's how to overcome this problem.

- 1 Increase all counters to a number that you know will be above the maximum required.

By increasing the counters to 100, your program will be capable of processing details for up to 100 students.

Enter:

```
200 FOR R = 1 TO 100 STEP 1
330 FOR R = 1 TO 100 STEP 1
```

- 2 Enter a conditional statement inside each loop, after the **GOSUB** statement.

```
220 IF N$ = "9999" THEN 245
350 IF N$ = "9999" THEN 375
```

- 3 Enter dummy data as shown:

```
9000 DATA "9999", "99", "99", "99"
9005 DATA "9", "9", "9", "9"
```

When the dummy data is read, the program will branch to another statement.

By placing the dummy data statements at high statement numbers, you may insert additional data statements for additional students, if required.

**SAVE as STUDENT DETAILS**

**LIST and RUN**

Print the **LIST** and **RUN**

## Centring a heading

Instead of printing the heading at the left-hand margin, you may prefer to centre it over the table.

Assume that you want to print on an 80 space line.

You could:

- 1 Count the characters in the heading (PERSONAL DETAILS = 16)
- 2 Subtract that number from the line length ( $80 - 16 = 64$ )
- 3 Halve that number ( $64/2 = 32$ )
- 4 Print heading at **TAB** position 32.

However, the system will do all of this for you, by using the:

**LEN** (length) statement (to determine the number of characters in the variable)

**INT** (integer) statement (to return a whole number)

Enter these statements:

```
170  GOSUB 650
290  GOSUB 650
650  REM  SUBROUTINE TO CENTRE HEADINGS
660  L =  LEN (H$)                (refer step 1 above)
670  T =  INT (80 - L) / 2        (refer steps 2 and 3 above)
680  PRINT  TAB ( T ); H$        (refer step 4 above)
690  RETURN
```

Delete statements 180 and 300

**SAVE as STUDENT DETAILS**

**LIST and RUN**

Print the **LIST** and **RUN**

## Using the **LEFT\$, MID\$, RIGHT\$** statements

With these statements, you may select specified characters from a variable to assist your programming.

eg **X\$ = "EXAMPLE"**

<b>L\$ = LEFT\$(X\$,2)</b>	returns the first two characters	<b>EX</b>
<b>R\$ = RIGHT\$(X\$,3)</b>	returns the last three characters	<b>PLE</b>
<b>M\$ = MID\$(X\$,2,4)</b>	returns four characters, starting with the second character	<b>XAMP</b>

When you run your program, there may be occasions when you do not want to print both lists. The statements below will give you that choice.

```
130 PRINT "DO YOU WANT TO PRINT PERSONAL DETAILS?":  
    PRINT  
140 GOSUB 600  
150 IF LEFT$(Y$,1) = "Y" THEN 160  
155 GOTO 250  
250 PRINT "DO YOU WANT TO PRINT STUDENT ENROLMENT?":  
    PRINT  
260 GOSUB 600  
270 IF LEFT$(Y$,1) = "Y" THEN 280  
275 GOTO 375  
600 REM SUBROUTINE FOR PRINTING CHOICE  
610 PRINT "ENTER YES OR NO AND PRESS RETURN": PRINT  
615 Y$ = "N"  
620 INPUT Y$  
630 RETURN
```

**SAVE as STUDENT DETAILS**

**LIST and RUN**

Print the LIST and RUN

This is a useful function — when the program is run, some operators would type YES, but others might type only Y. This example covers both types of input.

## Using the VAL statement

The VAL statement will convert a string variable into a numeric value. If you want to print the actual age of students, you could do this by comparing the student's year of birth with the current year.

To convert sections of string variables for birthdate (B\$) and current year (D\$) to numeric values.

**B = VAL(RIGHT\$(B\$,2))**      Returns year of birth

**D = VAL(RIGHT\$(D\$,2))**      Returns current year

To find the age of a student:

**A = D - B**

Enter these statements:

```
380 PRINT "DO YOU WANT TO PRINT STUDENT AGES?": PRINT  
390 GOSUB 600  
400 IF LEFT$(Y$,1) = "Y" THEN 410  
405 GOTO 9999
```

cont.

```

410 H$ = "STUDENT AGES"
420 GOSUB 650
425 PRINT : PRINT D$: PRINT
430 PRINT "NAME", "BIRTHDATE", "AGE", "ADDRESS"
435 PRINT
440 RESTORE
450 FOR R = 1 TO 100 STEP 1
460 GOSUB 700
470 IF N$ = "9999" THEN 9999
480 B = VAL ( RIGHTS$ (B$,2))
490 D = VAL ( RIGHTS$ (D$,2))
500 A = D - B
510 PRINT N$,B$,A,A$
520 NEXT R

```

**SAVE** as STUDENT DETAILS

**LIST** and **RUN**

Print the **LIST** and **RUN**

**Notes:**

- a** Keep this program on disk or cassette, as you will be working with it again in the next chapter.
- b** You could also compare the current month with the birth month, to get a more accurate age, by using the **MID\$** statement.
- c** Students' marks could be entered as data (instead of **E** or **N**) to obtain a printed list of actual results.

# 15 Arrays

An **array** is simply a list that stores the values for a number of variables.

Using the data from the previous chapter, the array (N\$) could be represented as a **one-dimensional** array.

## One-dimensional array

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Name	Address	Phone	Birthdate	Class	English	Maths	Science

The array N\$ now has eight columns, called **elements**, and they are referred to by the number of the element, eg

Name                      N\$(1)

Address                   N\$(2)

Phone                     N\$(3)

Birthdate                N\$(4)

Class                     N\$(5)

English

Maths

Science

Can you  
identify  
these  
elements?

If you want to use an array to **READ** and store all the information for one student:

**FOR E = 1 TO 8**            (number of elements)

**READ N\$(E)**            (to read element **E**)

**NEXT E**            (increase **E** by 1)

These statements form a loop. The first time through the loop, **E = 1** so the first data item (name) will be read and stored as **N\$(1)**. The second time through the loop, **E = 2** so the second data item (address) will be read and stored as **N\$(2)**.

This loop will continue until all data has been read and stored under its own variable name.

## Two-dimensional array

If you want to read and store data for more than one student, you must use a two-dimensional **array** (sometimes called a **matrix**).

This diagram represents the two-dimensional array (**N\$**).

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
	Name	Address	Phone	Birthdate	Class	English	Maths	Science
Row 1	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
Row 2	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8
Row 3	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8

All details for student 1 are stored in row 1.

All details for student 2 are stored in row 2.

All details for student 3 are stored in row 3.

Details	Student 1	Student 2	Student 3
Name	N\$(1,1)	N\$(2,1)	N\$(3,1)
Address	N\$(1,2)	N\$(2,2)	N\$(3,2)
Phone	N\$(1,3)	N\$(2,3)	N\$(3,3)
Birthdate	N\$(1,4)	N\$(2,4)	N\$(3,4)
Class	N\$(1,5)		
English	N\$(1,6)		
Maths	N\$(1,7)		
Science	N\$(1,8)		

Can you complete the missing variables?

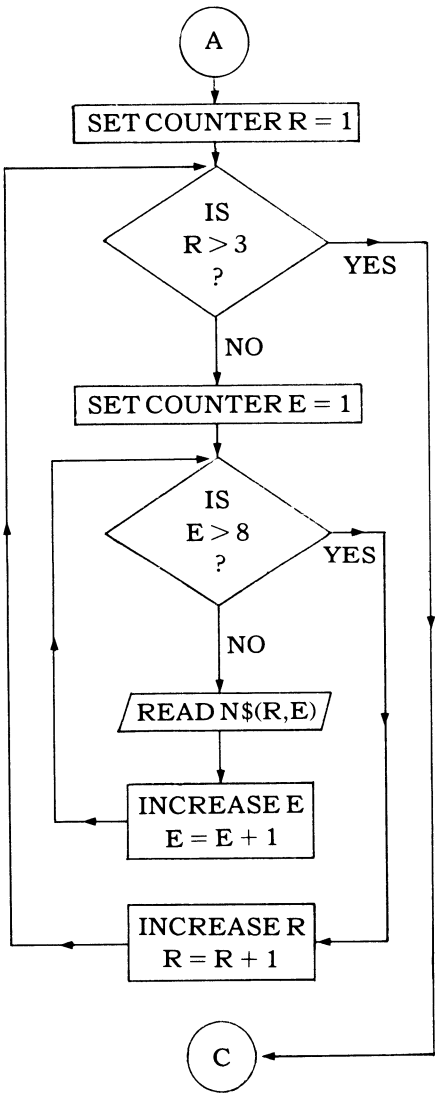


# Nested loop

To use a two-dimensional array to read and store information for more than one student, you will need to used nested loops, ie one loop which works inside another, eg

```
FOR R = 1 TO 3                (number of rows)
FOR E = 1 TO 8                (number of elements)
READ N$(R,E)                 (read row R, element E)
NEXT E                        (increase E by 1)
NEXT R                         (increase R by 1)
```

The nested loop is represented in a flowchart as:



The inner loop (E), which reads the elements, is nested inside the outer loop (R), which reads the rows.

The two loops must never cross each other, as such a condition would stop a program.

To adjust your program **STUDENT DETAILS** to use a two-dimensional array:

**1 Load STUDENT DETAILS**

**2 Save under another name, eg SAVE ARRAYS**

This will allow you to keep one program 'STUDENT DETAILS' and a separate program 'ARRAYS'.

**3 Delete these statements:**

40 to 100	variables description
210, 220, 230	Portion of first loop
320	RESTORE statement
340, 350, 360	portion of second loop
440	RESTORE statement
460, 470, 480, 510	portion of third loop
700, 710, 720	subroutine to read DATA

**4 If the two-dimensional array contains more than ten rows, then you must use a DIMENSION statement to set the size of the array, ie the maximum number of rows, and the maximum number of elements.**

For your program, you do not need a DIMENSION statement, but you may insert one for practice.

**ENTER**

```
5 DIM N$(100,8)
```

**5 Enter these statements to identify the variables:**

```
35 REM NAME N$(R,1)
40 REM ADDRESS N$(R,2)
45 REM PHONE N$(R,3)
50 REM BIRTHDATE N$(R,4)
55 REM CLASS N$(R,5)
60 REM ENGLISH N$(R,6)
65 REM MATHS N$(R,7)
70 REM SCIENCE N$(R,8)
```

**6 Enter these statements to READ and store all DATA:**

```
75 FOR R = 1 TO 100 STEP 1
80 FOR E = 1 TO 8 STEP 1
85 READ N$(R,E)
90 IF N$(R,1) = "9999" THEN 110
95 NEXT E
100 NEXT R
```

**7** Enter these statements to **PRINT** details for 'personal details':

```
210 IF N$(R,1) = "9999" THEN 245
220 PRINT N$(R,1),N$(R,3),N$(R,4),N$(R,2)
```

**8** Enter these statements to **PRINT** details for 'student enrolment':

```
340 IF N$(R,1) = "9999" THEN 375
350 PRINT N$(R,1),N$(R,5),N$(R,6),N$(R,7),N$(R,8)
```

**9** Enter these statements to **PRINT** details for 'student ages':

```
460 IF N$(R,1) = "9999" THEN 9999
480 B = VAL ( RIGHT$ (N$(R,4),2))
510 PRINT N$(R,1),N$(R,4),A,N$(R,2)
```

**SAVE** as **ARRAYS**

**LIST** and **RUN**

**PRINT** the **LIST** and **RUN**

# 16

## Colour and graphics

The colour, characters and graphics on the Commodore 64 are controlled by a single integrated circuit called the 6566/7 Video Interface Chip (advanced version of the VIC chip). It can be programmed to produce a variety of effects.

- Simple graphics are easily produced by using the graphics symbols which appear on the keyboard.
- The colours of any printed character are easily changed by holding down the CTRL or the C= key and pressing a COLOUR key and then typing the character.
- Screen and border colours are easily changed.
- Multicolour characters and high resolution plotting are available.
- Games paddles, joysticks and a light pen can be used.

To print characters, the Commodore 64 accesses a 'character generator' ROM. This ROM contains the pattern of dots which make up each character. One of the most important features of the Commodore 64 is that users can define their own set of characters. This allows creation of new characters, anything from foreign alphabets to special graphics characters such as 'space invader monsters', chemical symbols, music signatures etc. By setting the Commodore 64 to a 'bit mapped mode' (where you can turn any dot on or off), high resolution plotting (a 320 by 200 grid) or 4-colour plotting (160 by 200) can be achieved to draw multicolour lines, circles and graphs.

### Sprites

Sprites or movable object blocks are images of  $24 \times 21$  dots which can appear on the screen independently of the main screen image. For example, a sprite plane can fly across the screen, or a sprite figure can

walk across a landscape behind buildings (where it disappears) and in front of trees. You may have up to eight sprites on the screen at one time. Each can be moved independently and they can be checked for collisions with each other or the main screen image. Some of these features will be covered later but for more detailed discussions, together with worked examples, see the suggested reading at the end of this chapter.

Some of the more interesting colour and graphics programming is time consuming. There are no direct BASIC commands and you must fill particular memory locations with the required settings. If you wish to exploit these features more fully, there are good 'character editor' and 'sprite editor' programs available from Commodore in 'public domain' software. These do the hard work for you.

## How to enter programs in this section

Because of the number of special characters used in the Commodore 64, program listings often become difficult to read. For example, if you type a quote then press the CLR key to clear the screen, a reverse fielded 'heart' character appears. As we shall see below, this is a powerful programming feature of the Commodore 64. However, as a result of this, direct program listings can contain a large number of 'strange characters' which do not appear on the keyboard. To make the programs more readable we have adopted the following conventions.

- 1 All keys are referred to by the character appearing on the top face of the key with the exception of the colour keys, which are printed within square brackets as **BLK**, **WHT**, **RED** etc, and the cursor control keys, which are printed as **CLR**, **HOME**, **DOWN**, **UP**, **LEFT**, **RIGHT**.
- 2 Any characters enclosed in square brackets [ ] or less than and greater than symbols < > require an action by the user. **Do not type these symbols.**
- 3 Characters enclosed within square brackets are either colour keys or require the user to hold down either the shift or control keys, **then** type the enclosed characters.
- 4 Characters enclosed within [ < > ] require the user to hold down the Commodore key (C=) **then** type the enclosed character.

Examples of these conventions are:

Listed as	Key/s to press
[DOWN]	cursor down
[UP]	shift and cursor up
[RIGHT 5]	cursor right five times
[CLR]	shift and CLR
[HOME]	HOME

Listed as	Keys to press
[BLK]	CTRL and BLK
[RVS]	CTRL and RVS ON
[OFF]	CTRL and RVS OFF
[A]	shift and A (produces spade character)
[–]	shift and –
[<BLK>]	Commodore and BLK
[<A>]	Commodore and A (produces character on front left of key)

If more than one of the above are used together they are separated by a comma

eg [CLR, DOWN 5, RIGHT 3]

## Colour

There are 16 different colours available for either the border, screen or character colours.

Character colours are changed by holding down the CTRL or C= and pressing one of the COLOUR keys.

Any combination of screen and border colours can be set by putting the colour number (0 – 15) into the particular colour register (memory location) of the chip.

Not all of these go together well. You should construct a chart of character versus screen colours to find the best matches.

Screen and border colours are changed by:

Screen colour	POKE 53281,C (C = 0 – 15)
Border colour	POKE 53280,C (C = 0 – 15)

To see the total combinations of screen and border colours:

NEW

```

10 REM CHANGE SCREEN/BORDER COLOURS
20 SC = 53281 : BC = 53280
30 FOR S = 0 TO 15
40 POKE SC, S
50 FOR B = 0 TO 15
60 POKE BC, B
70 PRINT "[CLR, DOWN 3] BORDER = "B
80 PRINT "SCREEN = "S
90 REM TIMING LOOP
100 FOR T = 1 TO 500 : NEXT
110 NEXT B
120 NEXT S
RUN

```

To return the screen to normal:

Press RUN/STOP and RESTORE

Changing the screen and border colours can be very useful in programming to indicate an error or some special condition (or just for fun).

## Simple graphics

One of the two character sets of all Commodore computers contains 63 special graphics symbols (plus 63 with the colour reversed). These are accessed in the normal mode in two ways. Firstly, the graphic symbol on the left of the key is obtained after holding down the C= key and the graphic symbol on the right with the shift key. Reverse field characters are obtained by first pressing the RVS ON key, then as above. Press RVS OFF to turn reverse characters off.

Any of these can be drawn on the screen by simply printing them as normal characters included within quotes as with normal strings. The colour of each can be changed with the CTRL or C= and COLOUR keys as usual.

## Using control characters

- 1 Anything enclosed by square brackets is either a graphics (use shift or C= key) or a cursor control character. **Do not type the square brackets.**
- 2 Any character, including cursor controls and colour controls, may be printed within quotes. When a string containing cursor or colour control is printed, these behave **exactly** as in the direct mode to either move the cursor or change to a new colour. For example:

NEW

```
10 PRINT "[CLR, DOWN 10]"
```

means that after you type the opening quotation mark, you then press shift and CLR, then press the cursor down ten times, then type the closing quotation mark.

```
20 PRINT "[CYN]HELLO I'M HERE"
```

The action indicated by the square brackets is to press CTRL and CYAN keys.

RUN See what happens?

Press RUN/STOP RESTORE for normal colour.

- 3 Once a quotation mark (") has been typed (or the INSERT key has been used), the Commodore 64 is in the programmed cursor mode (PCM) where any cursor or colour control will simply produce a

'strange' reverse field character. This allows programming of these as above. However, once you type an opening quotation mark and are in PCM you now can't use the cursor! Try it!

To escape from PCM:

- a another quotation mark returns to normal mode.
- b RETURN enters the screen line into the computer.
- c shifted RETURN stops PCM and moves to next line — line is not entered into the computer therefore not changed. Use the cursor to edit.

## Programming screen positions

Part of the 'user friendliness' of the Commodore 64 is its logical and simple onscreen editing. This extends to simple graphics using the PCM mode.

Characters are printed at the cursor position. This can easily be controlled to allow printing anywhere on the screen at will.

It is done by having two strings of cursor controls (lines 1010 and 1020). We then select the appropriate left number of cursor movements to move to the position we want.

Two other things — in PCM a HOME [HOME] returns the cursor to the top left and a shift CLR [CLR] clears the screen.

```
50 REM RANDOM 40 BY 22 SCREEN PRINT
60 REM CLEAR SCREEN
100 PRINT"[CLR]"
110 N$="HELLO I'M HERE"
120 REM SELECT RANDOM VERT AND HORIZ
130 FORI=1TO50
140 V=1+RND(0)*22
150 H=RND(0)*39
160 GOSUB1000
170 PRINTN$
180 REM WAIT A BIT
190 FORT=1TO500:NEXT
195 GOSUB1000
196 REM ERASE MESSAGE
200 PRINT"
210 NEXT
220 END

1000 REM POSITION CURSOR
1010 H$="[LEFT 40]"
1020 V$="[HOME,DOWN 22]"
1030 PRINTLEFT$(V$,V);
1040 PRINTLEFT$(H$,H);
1050 RETURN
```



The action required in line 1010 is to press the cursor left key 40 times.  
The action required in line 1020 is to press HOME, then cursor down key 22 times.

*Exercise:* try changing the string N\$ to include a colour control, such as:  
110N\$ = "[CYN]HELLO I'M HERE"

*Exercise:* Try other strings and colours.

To draw anything on the screen at a particular position we can now use the routine above.

**Note:** Another method of positioning the cursor is to fool the Commodore 64 by changing its memory pointers, eg

Location	Points to
211	position of cursor on line
214	row where cursor lives - 1

Now replace lines 1010 - 1040 with

```
1020 POKE 214, V : PRINT : POKE 211, H
```

## Line drawing

To draw lines, we select the line character from the graphics set and print it with the corresponding cursor controls.

For example, a horizontal line:

```
FOR I = 1 TO 40 : PRINT "[C]"; : NEXT
```

A vertical is as easy, although it is more involved:

```
PRINT "[-,DOWN,LEFT]"
```

prints a vertical bar and moves the cursor to directly under the line. So we repeat the sequence twenty times as:

```
FOR I = 1 TO 20 : PRINT "[-,DOWN,LEFT]"; : NEXT : PRINT
```

Diagonals are done by printing the character, then moving down one line, as:

```
FOR I = 1 TO 20 : PRINT "[M,DOWN]"; : NEXT : PRINT
```

This type of simple graphics is shown in the following routine which draws a border and a message in conjunction with the position subroutine above (lines 1000 - 1050). The program also uses colour changes.

```

100 REM SET BORDER COLOUR TO RED
110 POKE 53280,2
120 REM DRAW BORDER
130 PRINT"[CLR]";
140 FORI=1TO39
150 PRINT"[+]";

```

Remember ↑ Means shifted +

```

160 NEXT
170 PRINT
180 FORI=1TO22
190 PRINT"[<Q>]";TAB(38);"[<W>]"
      ↑ Means C= and Q
200 NEXT
210 FORI=1TO39
220 PRINT"[+]";
230 NEXT
240 PRINT
250 REM NOW POSITION CURSOR
260 H=10:V=10:GOSUB1000
270 C$="[CYN]"
280 PRINTC$;"A. BORDER"
290 H=10:V=12:GOSUB1000
300 PRINT"39*22 NOT OUT"
320 REM WAIT FOR A WHILE
330 FORT=1TO5000:NEXT
340 REM RESET BORDER COLOUR
350 POKE53280,14
360 END

```

Remember to re-enter the subroutine at line 1000 – 1050

## Picture drawing

These graphics symbols can be readily used to draw simple pictures of limited resolution.

Press CLR

*Exercise:* By using the cursor controls and the graphics symbols, see if you can draw a picture of a house (leave 5 spaces on the left of the screen). It takes a bit of practice!

When you decide you have something you like:

HOME the cursor and

Move down the left of the screen, type line numbers and PRINT"

Then RETURN after each line.

You now have a program, which will print your picture.

*Mystery puzzle:* If you were not successful with the exercise above, try to follow this one.

**Note:** Most characters are to be entered with the shift key to get the graphics, except for those enclosed with symbols <>. **Do not type the <> characters.**

```
300 PRINT"      NM"
310 PRINT"      N  M"
320 PRINT"      N   M"
330 PRINT"      N    M      <A>****<S>"
340 PRINT"      N      M      BVVVVE"
350 PRINT" <TTTTTTTT>P      BVVVVE"
360 PRINT" <MDDDDDM>      <O>****<X>"
370 PRINT" <MDDDDDDM>      B"
380 PRINT" <M>  <@#@> <M>      B"
390 PRINT" <M>  <G> <G> <M>      B"
400 PRINT" <M>  <G> <G> <M>      B"
410 PRINT" <TTTTTTTTTT>      B"
420 PRINT"      B"
430 PRINT"      B"
440 PRINT" <OBBBBBBBBBB>      B"
450 PRINT" <OBBBBBBBBBB>      U<E>I"
460 PRINT"      <Z>*<X>"
```

## Screen and colour memory

Like most microcomputers, the screen is actually a RAM area of memory and instead of printing characters to the screen you may simply load the character code into the screen RAM.

This screen RAM starts at memory location 1024 and occupies 1000 consecutive memory locations (25 rows of 40 columns).

Screen character RAM starts at 1024

For example, to put A into the top left of the screen (see references for character codes or try any number and see what happens):

POKE 1024,1 (1 is the screen code for A)

Do you see the A? No — you don't!

Now move the cursor up to the top left and you will see the A flashing.

What has happened is that we have put an A in the correct position but its colour is the same as the background so we cannot see it. The second part of this operation requires putting a colour (0 — 15) into the screen colour RAM. This is also 1000 bytes long and is at 55296.

Screen COLOUR RAM starts at 55296

Now choose a colour for our A, eg CYAN (= 3, the colour code is one less than the number on the colour key) and put into corresponding colour position by

```
POKE 55296,3
```

We now see a CYAN A.

It's easy to have fun with the colour memory. For example:

```
100 REM RANDOM CHARACTER COLOURS
110 FOR I=0 TO 1023
120 SC=55296+I
125 REM SELECT COLOUR 0-15
130 C=INT(RND(0)*15)
140 POKE SC,C
150 NEXT
```

Now type a screen full of rubbish and RUN the program.

*Exercise:* Add line 115 POKE 1024+i, 160

By poking characters to the screen (+ colour) RAM, simple game programs can be developed. The screen is a 40 column (0 — 39) by 25 row (0 — 24) grid.

To move right, add 1 to the location

Left	subtract 1
Down	add 40
Up	subtract 40

But you must check that the new memory location is still in the screen RAM area between 1024 and 2023.

For example, this program allows you to move the 'heart' graphics character (shifted S) around the screen with the cursor controls.

```
100 REM SCREEN CHARACTER CODE
110 CH=83
120 REM PUT AT MIDDLE OF SCREEN
130 P=500
140 POKE 1024+P,CH
150 REM COLOUR = CYAN
160 POKE 55296+P,3
170 GETA$:IFA$="" THEN 170
175 POKE 1024+P,32
180 IFA$=CHR$(29) THEN P=P+1
190 IFA$=CHR$(157) THEN P=P-1
200 IFA$=CHR$(17) THEN P=P+40
210 IFA$=CHR$(145) THEN P=P-40
215 REM CHECK IF WITHIN LIMITS
220 IF P>1000 THEN P=P-1000
230 IF P<0 THEN P=P+1000
240 GOTO 140
```

**Note:** The GET instruction gets a key from the keyboard. If no key has been pressed it gets nothing, ie "".

# More difficult graphics

## Character generator

As discussed before, the Commodore 64 uses a character generator ROM which stores the actual pattern of dots which make up each character.

Let's have a closer look at how this is actually done and then make some new characters of our own.

If you look closely at the screen, you will see that each character is made up of an 8 by 8 dot pattern. Each of these dots is called a **pixel**. (Think of them as small light bulbs.)

Type a capital A and look closely. It is made up by:

COLUMN									ROW VALUE
	7	6	5	4	3	2	1	0	
ROW									
0				*	*				= 00011000 = 24
1			*	*	*	*			= 00111100 = 60
2		*	*			*	*		= 01100110 = 102
3		*	*	*	*	*	*		= 01111110 = 126
4		*	*			*	*		= 01100110 = 102
5		*	*			*	*		= 01100110 = 102
6		*	*			*	*		= 01100110 = 102
7									= 00000000 = 0
	↑	↑	↑	↑	↑	↑	↑	↑	
		64		16		4		1	
	128		32		8		2		

If we allow each pixel (or light bulb) to be either ON or OFF (as only light bulbs can) and we say that ON = 1 and OFF = 0, we can then convert each row into a pattern of 1's and 0's as on the right hand side.

This is a **binary** number. Binary is a number system wherein each column represents a power of 2. (Hence the name binary.) Remember

that in our 'normal' decimal system, each column represents a power of 10.

So the numbers on the bottom of each column represent the power of two for that column. The first row is thus made up of:

$$0*128 + 0*64 + 0*32 + 1*16 + 1*8 + 0*4 + 0*2 + 0*1 = 24$$

The other rows are added up the same way to get the corresponding 'decimal' value for the row.

Microcomputers work in binary numbers and each dot or pixel above is called a **bit**. To make life simpler, bits are grouped into blocks of 8 (as above). This block is called a **byte**.

Each memory location within the computer's ROM and RAM holds one byte.

Thus, to store our character pattern for 'A' requires 8 bytes or 8 memory locations. And to store the **two** complete character sets (uppercase/graphics and lowercase/uppercase) of 255 characters, each 8 bytes long, takes 4096 bytes (or 4 K) of ROM and starts at memory location 53248.

In our character generator ROM we would then find 8 successive bytes containing the numbers 24, 60, 102, 126, 102, 102, 102 and 0, giving our pattern for 'A'.

Where is all this heading?

## Programmable characters

The important feature of the Commodore 64 is that the user can tell the computer to get the pattern for each character **not** from its ROM area but from an area in RAM. And since RAM memory can be changed by the user, a completely new character set can be 'user defined'.

The Commodore 64 tells itself where to get the character pattern with a pointer in location 53272. When you press the C= and shift keys to change between sets, what is actually happening is that this pointer is changing between the two 2 K areas of ROM. As:

```
POKE 53272,21  uppercase/graphics
POKE 53272,23  lowercase/uppercase
```

**Beware:** This location also controls the screen memory location. Try some other value and the screen will turn to rubbish.  
(Use RUN/STOP and RESTORE for normal.)

In this same manner, we can change the pointer to our user defined character set in RAM.

We do not need to write a totally new set of 255 characters, however. If the old set (or the part we wish to use) is transferred to the RAM area, we may selectively change only the ones we wish.

The user RAM area occupies locations 2048 — 40959. We have to be careful where we put our new character set so that the area is not used by BASIC.

In this example, the new character set will be put at location 12288 up.

To move the ROM character set starting at 53248 to our 12288 RAM is somewhat complicated by the way the Commodore 64 works. However, the following will move 128 characters from ROM to RAM and can be easily adapted.

#### NEW

```
102 PRINT"[CLR,DOWN 5] PLEASE WAIT"  
105 IN=56333:GK=12:B=GK*1024:CR=53248:CP=53272
```

Our new characters start at b = 12288

```
110 POKEIN,127  
115 POKE1,51  
120 FORI=0TO128  
125 FORJ=0TO7  
130 POKE B+I*8+J,PEEK(CR+I*8+J)  
135 NEXT: NEXT  
140 POKE1,55:POKEIN,129
```

**Note:** Lines 110, 115 and 140 **must** be used in this order.

Now to change the character pointer to our new RAM area at 12288.

```
150 POKECP,(PEEK(CP)AND240)+GK
```

#### RUN

Nothing has changed! Why?

Because the Commodore 64 is still using the **same set** of characters but from a different area of memory.

To change a character pattern, we must first find out where in our RAM area it is stored. This is done using the screen character code. This is its position in the table of characters. Refer to the User Guide for details.

Let's use the A. For the alphabet, we can use the formula.

```
PRINT ASC("A")—64
```

Answer 1

Remember that each character uses 8 bytes so our pattern for A starts at location

```
START 12288 + 8 * 1 = 12296
```

and extends for 8 bytes.

**Exercise:** To create a new character pattern:  
draw a new character pattern on the 8 by 8 grid.

ROW	COLUMN								Value
	7	6	5	4	3	2	1	0	
0									= ?
1									= ?
2									= ?
3									= ?
4									= ?
5									= ?
6									= ?
7									= ?

Add the rows as we did above to get 8 numbers.

Put those numbers into our 8 successive locations using POKE statements such as:

POKE 12288 + row number, row value

Once this is done, every time you press an A character you will get your new patterns.

If you get into trouble by doing this, try pressing RUN/STOP and RESTORE. If that does not work turn off and on again.

Still confused? The following program will change any three characters you choose to give some interesting patterns.

Add the following lines to the above program

```

100 REM THREE NEW CHARACTERS
101 POKE53280,2
160 POKE53280,14
175 FOR T=1T03
180 INPUT"OLD CHAR";C$
190 F=B+(ASC(C$)-64)*8
200 FORI=0T07
210 READA
220 POKEP+I,A
230 NEXT
235 NEXT
240 DATA0,24,60,126,255,24,36,6
250 DATA24,20,20,18,48,112,96,0
260 DATA60,66,165,129,165,153,66,60

```



Now whenever you type the old character, your new pattern will appear.

## High resolution graphics

High resolution (or bit mapped) graphics are achieved in a similar manner to user defined characters. But each dot on the screen (pixel) corresponds to one bit in memory. Since this needs 64000 bits (1000 characters \* 8 \* 8), we need to use 8000 bytes of memory. We then can selectively and dynamically (during the program operation) create new characters which will light whichever pixel we desire for the line or graph.

The Commodore 64 will generate a two colour 320 by 200 or a four colour 160 by 200 grid.

Since the programming of high resolution graphics has some pitfalls, you should consult the references for details. The following program will demonstrate the two colour mode and draw the graph of a sine wave on a 320 by 160 grid. It takes a while; be patient! (The **bit** map is located at 12288 up.)

Example:

```
100 REM COSINE CURVE
110 GK=8:B=GK*1024:CP=53272:BM=53265
120 REM SET CHARACTER POINTER TO RAM
130 POKECP,PEEK(CP)OR GK
140 REM SET BIT MAP MODE ON
150 POKEBM,PEEK(BM)OR 32
160 REM CLEAR BIT MAPPED RAM
170 FOR I=BTOB+7999:POKEI,0:NEXT
180 REM SET SCREEN TO BLACK/YELLOW
190 FOR I=1024TO2023:POKEI,7:NEXT
200 REM CALCULATE X,Y AND PLOT
210 FOR X=0TO319STEP.5
220 Y=INT(90+80*COS(X/10))
230 CH=INT(X/8)
240 RO=INT(Y/8)
250 LN=YAND7
260 BY=B+RO*320+8*CH+LN
270 BI=7-(XAND7)
280 POKEBY,PEEK(BY)OR(2*BI)
290 NEXT
300 POKE1024,16
310 GOTO310
320 REM RUN/STOP RESTORE FOR NORMAL
```

## Sprites

Constructing sprites is like constructing large characters. Two colour sprites are made on a 24 by 21 grid. A simple example is given below but it

is better to use a sprite editor program. (Four colour sprites are slightly more complicated to program.)

You can in fact use almost any number of sprites. However, more than 3 require complicated programming. The following method may be used relatively simply to create three sprites only.

In the same way as programming characters, the dot pattern is worked out but on this 24 by 21 grid, instead of an 8 by 8 grid. However, with sprites, each row is made up of 3 bytes, ie there are three numbers for each row, and there are 21 rows. Each byte number is the sum of its 8 dot values (as above).

### Sprite constructing grid

	Byte 1								Byte 2								Byte 3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ROW 0																								
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20																								

The calculated 63 numbers for your pattern must now be placed in memory in order for the Commodore 64 to display it on the screen. We will put them from location 832 up by poking the numbers into memory. Two more may be put at  $832 + 64$  and  $832 + 128$ . **No more can be placed here;** it upsets BASIC!

For 63 locations:

POKE  $832 + \text{row} * 3 + \text{byte}$ , byte number

(Rows 0 — 20 Bytes 0 — 2)

It is better to put numbers in a DATA statement and READ and POKE in a loop.

For example:

```
FOR I = 0 TO 62 : READ A : POKE 832 + I, A : NEXT
```

Now that the pattern is set, there are various sprite pointers which enable the Commodore 64 to use it. These are:

### 1 Sprite image pointers

Location	2040	2041	2042	2043	2044	2045	2046	2047
Sprite No	0	1	2	3	4	5	6	7

Each sprite uses 64 bytes (63 for image + spare). This pointer sets the image memory location in 64 byte blocks.

For image at 832, value is 13. POKE 832, 13

### 2 Sprite control registers (16)

Control the sprite screen position (0 — 343 horizontal, 0 — 249 vertical) from top left of screen (0,0) with reference to the top left of the sprite. **Not all** positions will be on the screen.

Sprite 0 53248 x co-ord. (0 — 255)  
53249 y co-ord. (0 — 255)

**In general use** V = 53248 and registers are:

x co-ord. V + sprite no (SN) × 2  
y co-ord. V + 1 + SN × 2

**Note:** You must also set “m.s.b.” of X if using X > 255

If X > 255 then X = INT(X/255) and POKE V + 16, PEEK(V + 16)  
OR (2 SN)

If X < 255 POKE V + 16, PEEK(V + 16)  
AND (255 - SN)

ie Sprite 2 POKE V + 4, X  
POKE V + 5, Y

### 3 Make sprite appear register

Turn sprite on POKE V + 21, PEEK(V + 21) OR (2 SN)  
Turn sprite off POKE V + 21, PEEK(V + 21) AND (255 - 2 SN)

ie Sprite 2 on POKE V + 21, PEEK(V + 21) OR 4  
off POKE V + 21, PEEK(V + 21) AND 251

### 4 Sprite colour registers — one for each

POKE V + 39 + SN, colour of 0-15

ie Sprite 2 colour CYAN POKE V + 39 + 2, 3

## 5 Expand sprite by 2 horizontally (ie put twice as large pattern on screen)

Expand      POKE V + 29, PEEK(V + 29) OR (2 SN)

Normal      POKE V + 29, PEEK(V + 29) AND (255 - 2 SN)

ie Sprite 2 x expand      POKE V + 29, PEEK(V + 29) OR 4

## 6 Expand sprite vertically

Expand      POKE V + 23, PEEK(V + 23) OR (2 SN)

Normal      POKE V + 23, PEEK(V + 23) AND (255 - 2 SN)

ie Sprite 2 y expand      POKE V + 23, PEEK(V + 23) OR 4

It takes some getting used to, but patience brings rewards. See the User Guide for more details.

Other sprite options: multicolour; check for collisions between sprites or sprite-screen; set priority between sprites; or priority to screen.

# Demonstration program

Instead of calculating the numbers for the sprite pattern, it is more sensible to get the computer to do the hard work.

This program:

- Creates three sprites from the patterns in the data statements.
- Calculates the numbers and pokes them into the memory. (An A in the data statement is taken as a dot (1), a space as a space (0)).
- After each pattern is fully in the memory, it expands the sprite in both x and y.
- After all three are in the memory, sprites are moved about simply and randomly expanded in the centre of the screen.
- Sprites of lower number move over higher number.
- Display priority of sprite 2 is set so that it moves behind the screen image.

```
100 REM THREE SPRITE GENERATOR
110 V=53248:REM MAIN SPRITE POINTER
140 REM CALCULATE AND STORE 3 IMAGES
150 FOR SN=2 TO 0 STEP -1:RESTORE:IMAGE=832+SN*64
160 REM CLEAR IMAGE RAM
170 FOR I=IMAGE TO IMAGE+63:POKE I,0:NEXT
180 REM SET IMAGE POINTER TO DATA BLOCK
190 POKE 2040+SN,IMAGE/64
200 REM POSITION SPRITE X AND Y ON SCREEN
210 POKE V+SN*2,50+SN*20
220 POKE V+1+SN*2,50+SN*20
230 REM TURN ON SPRITE SN
240 POKE V+21,PEEK(V+21)OR(2*SN)
250 REM SET COLOUR OF SPRITE SN
```

```

260 POKE V+39+SN,16+SN
270 REM GET 3 NUMBERS FOR EACH OF 21 ROWS
280 FOR ROW=0TO20
290 READD$:PRINTD$;
300 FOR I=0TO2:N=0
310 FOR BIT=7TO0STEP-1
320 REM CALCULATE NUMBER
330 IFMID$(D$,1+I*8+7-BIT,1)="A" THENN=N+2*BIT
340 NEXT BIT
350 REM PUT NUMBER INTO IMAGE MAP
360 POKE IMAGE+ROW*3+I,N:PRINTN;
370 NEXT I :PRINT
380 NEXT ROW
390 REM EXPAND SN IN BOTH X & Y
400 POKE
V+23,PEEK(V+23)OR(2*SN):POKEV+29,PEEK(V+29)OR(2*SN)
410 NEXT SN
420 REM MOVE SPRITES
421 PRINT"[CLR,DOWN 4,RVS]";FORI=1TO40*6:PRINT"<0>";
:NEXT
422 REM SPRITE 2 UNDER SCREEN
423 POKE53275,2
430 POKEV+23,0:POKEV+29,0
440 S=255:FORI=1TO255
450 POKEV,I:POKEV+1,I
470 POKEV+2,S-I:POKEV+3,S-I
490 POKEV+4,I:POKEV+5,S-I
510 IFI<>128THEN550
520 REM RANDOMLY EXPAND
530 IFRND(0)>.2THENPOKEV+23,RND(0)*8:POKEV+29,RND(0)*8
550 NEXT:GOTO430
590 REM "765432107654321076543210"
600 DATA" AAA A AAA "
601 DATA" AAAAA A AAAAA "
602 DATA" AA AAAAA AA "
603 DATA" AA A A A AA "
604 DATA" AA A AA "
605 DATA"AA A AA "
606 DATA"A A A "
607 DATA" A "
608 DATA" AAAAAA A AA "
609 DATA" AAAAAA A AA "
610 DATA" AA AA A AA "
611 DATA" AA A AA "
612 DATA" AA A AA "
613 DATA" AAAAAA A AA AA "
614 DATA" AAAAAA A AA AA "

```

cont.

```

615 DATA"  AA   AA  A  AA AA   "
616 DATA"  AA   AA  A  AAAAAAA "
617 DATA"  AAAAAAA A  AAAAAAA "
618 DATA"A AAAAAAA AAA   AA  A "
619 DATA"A          AAAAA  AA  A "
620 DATA"AAAAAAAAAAAAAAAAAAAAAA "

```

#### *Exercises:*

- 1** Design your own sprite in data statements.
- 2** Change sprite colours.
- 3** Change sprite movement.
- 4** Change expansion to x expand only.

## Books and suggested reading

These should be readily available from your local dealer, computer bookshop or newsagent.

- 1** *CBM 64 Programmers Reference Guide* by Commodore
- 2** *Compute!* Journal for progressive computing (US)
- 3** *Practical Computing* (UK)
- 4** *Commodore Computing International* (UK)
- 5** *Commodore User* (UK)

# 17

## Sound

Sound on the Commodore 64 is generated by an integrated circuit called the 6581 sound interface device or SID chip.

It is more powerful than the traditional music synthesisers of a few years ago and is capable of producing excellent music and other sounds in three part harmony. Some of its capabilities are:

- Volume control with 15 steps (48 db range)  
*Note:* The abbreviation **db** stands for decibel.
- The SID contains three independent oscillators or voices which each cover a 0 — 4K Hz range, giving almost eight octaves of usable notes.
- Each voice has four different waveforms (triangular, sawtooth, variable pulse and noise) which produce different harmonic structures.
- Each voice has a separate 'sound envelope generator' with independent control of:
  - a** attack rate 2ms — 8s
  - b** decay rate 6ms — 24s
  - c** sustain rate 0 — peak volume
  - d** release rate 6ms — 24s
- Voices can be linked (synchronisation) to obtain more complex harmonic structures such as vibrato and 'ring modulated' to produce bell and gong sounds.
- There are selectable filters between 30Hz — 12 KHz (12 db/octave) to produce low pass, band pass, high pass and notch filtering.

**All** of these can be changed during a program.

All three voices can be combined, allowing almost total control of the sound structure and permitting a good imitation of a wide range of musical (and non-musical) instruments.

However, there are no direct BASIC commands to control all these functions. It all has to be done by loading the SID's memory locations (registers) with the required numbers (with POKE statements).

Obviously, we can only cover simple sound applications. These are not difficult (they just require patience) and can allow you to easily explore some of the possibilities.

Consult the reference and user guides for more information.

Musicians should note that although the SID can be set for 'absolute pitch', there are at least two versions of the Commodore 64 in the world (US NTSC and European PAL) which run at different electrical frequencies. Most of the sound programs to date are set for relative pitch only and not absolute. However, the relative pitch produces a satisfactory sound.

For a correct table of notes, see Appendix 1.

## Programming music

To produce music or noise you must put the desired settings into the RAM memory of the SID. The steps are:

- 1 Select the overall volume with:

POKE 54296 with 0 — 15 (0 = off)

- 2 Select one of the voices (or up to 3)

Voice	Start Location
1	54272
2	54279
3	54286

- 3 Then set up each voice for the frequency, waveform and 'sound envelope' so as to play the notes. This can take up to seven steps for **each** voice, depending on the sound you want. These are done by loading the SID memory with:

POKE voice start location + function, setting

FUNCTION (F)	SETTING
0	frequency number low (FL)
1	frequency number high (FH) (found from table of notes)

For variable pulse waveform only (see later)

2	pulse width low (PL)
3	pulse width high (PH)

Waveform

	ON	WAVEFORM	OFF
4	17	triangle	16
	33	sawtooth	32
	65	variable pulse	64
	129	noise	128



Envelope generator

5 Attack x 16 + Decay

6 Sustain x 16 + Release

eg for voice 1

V = 54272

POKE V + F, SETTING

These should be set in the following order:

**a** volume (0 — 15)

**b** frequency (from table)

**c** A D S R (each 0 — 15)

**d** waveform (17, 33, 65, 129)

then **e** play note for desired time (by using a FOR/NEXT delay loop)

then **f** turn waveform off (16, 32, 64, 128)

Depending on the ADSR settings, in general a note will play until either the volume is turned off or the waveform is reset. You should play the note for a while then reset WAVEFORM to the OFF number above. This allows the note to die away slowly as controlled by the ADSR settings.

Obviously, in playing a sequence of notes (hopefully a tune!), the shorter the duration of each note, the faster the tempo.

**Don't panic!** It is not at all difficult. It just takes a bit of getting used to.

Before we look in detail at how each of these settings works, enter and run this program which plays the octave of notes.

100 REM SIMPLE OCTAVE

110 REM USING VOICE 1 AND TRIANGLE WAVE

Select voice and waveform:

115 V = 54272 : W = 17

Volume to maximum:

140 REM VOLUME ON

150 POKE 54296, 15

Set A D S R

170 REM SET A D S R

180 POKE V + 5, 7 : POKE V + 6, 183

Get notes from data statements:

210 REM READ AND PLAY NOTES

220 FOR I = 0 TO 7

230 READ FH, FL

POKE frequencies:

240 REM SET NOTE FREQUENCIES

250 POKE V, FL

260 POKE V + 1, FH

Set waveform:

270 REM USE TRIANGLE WAVE W = 17

280 POKE V + 4, W

Let note play for a while:

```
290 REM WAIT NOTE LENGTH
300 FOR T = 1 TO 500 : NEXT
```

Turn waveform off:

```
310 REM WAVEFORM OFF
320 POKE V + 4, W — 1
```

A little silence:

```
330 REM WAIT BETWEEN NOTES
340 FOR T = 1 TO 50 : NEXT
```

Get next notes:

```
350 NEXT
360 END
```

Data for notes C, D, E, F, G, A, B, C come from table 1.

```
370 DATA 17, 127, 19, 163, 22, 11, 23, 91
380 DATA 26, 55, 29, 109, 33, 7, 34, 254
```

Let's now look at how these settings created the sound.

#### 1 Set volume 0 — 15

POKE 54296, Volume (0 to turn off)

#### 2 The voice start location is chosen from table above. All other locations are this value plus the number in the left hand column above.

#### 3 Frequency numbers are chosen from the note table in Appendix 1; **or**, if you do not like them, choose a number between two; **or** a note of any frequency may be used as follows:

- Take the note "A" of 440 Hz

- Calculate the number:

$$N = \text{Freq} * 17.1196 \quad (440 * 17.1196 = 7533)$$

- Calculate the high and low frequency numbers by:

$$FH = \text{INT}(N/256) \quad (= 29)$$

$$FL = N - FH * 256 \quad (= 109)$$

- Then put these into 0 and 1 of the voice by:

POKE V, FL

POKE V + 1, FH

**Numbers must be less than 255.**

#### 4 Waveform is chosen from one of the four types.

ON	OFF	waveform	description
17	16	triangle:	only odd harmonics, mellow flute-like tone
33	32	sawtooth:	both odd and even harmonics, bright brassy tone

65	64	pulse:	variable square wave, adjustable harmonic content (timbre) by width of signal. <i>Note:</i> must set width in location 2 and 3.
----	----	--------	--

129	128	noise:	white noise for drums, gunshots, surf, wind, trains, etc.
-----	-----	--------	---

**Note:** The “off” value is the number less 1.

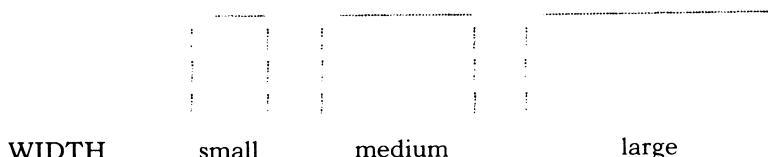
Select wave number and:

POKE V + 4, wave number

If pulse waveform:

Select width of pulse  $PW = 0 — 4095$  (0 = off)

This is the width of the rectangular pulse. The height is not adjustable, eg



Calculate high pulse number

$$PH = \text{INT}(PW/256)$$

Calculate low pulse number

$$PL = PW - PH * 256$$

POKE V + 2, PL

POKE V + 3, PH

- 5** Sound envelope generator consists of four parts — attack, decay, sustain and release.

When a sound is made, the volume (amplitude) changes with time. For example, when a drum is struck, the sound volume increases rapidly then dies away quickly. When an organ note is played, the volume increases quickly, holds as long as the key is pressed then dies away. When you play a tape backwards, the notes increase slowly in volume then drop away quickly. Different instruments give different patterns.

**Attack:** the initial rise in the sound volume.

**Decay:** the sound rises to a peak then drops to an intermediate level.

**Sustain:** the time the sound stays at this intermediate level.

**Decay:** the time for the sound to die away.

By setting these independently, the sound can be changed to resemble a range of instruments or even new ones.

A and D are set together, as are S and R. All of these can have value 0 (lowest) to 15 (highest).

To set attack and decay:

Select Attack 0 — 15

Select Decay 0 — 15

Calculate  $AD = A * 16 + D$

POKE V + 5, AD

To set sustain and release:

Select Sustain 0 — 15

Select Release 0 — 15

Calculate  $SD = S * 16 + D$

POKE V + 6, SD

**Exercises:** Try changing the above program.

- 1 Change waveform (W in line 200)
- 2 Change sound envelope (line 180)
- 3 Change frequency (in DATA lines 370, 380)
- 4 Change note timing in the delay loops (lines 300 and 340).

Try the following settings:

Instrument	A	D	S	R	WAVE
Violin	10	8	10	9	17
Drum	0	9	0	9	129
Piano	0	9	0	0	17
	0	9	2	11	17
Harpsichord	0	9	0	0	33
Organ	0	0	15	0	33
Accordian	12	0	15	0	17

**Remember:** The tone will change with the waveform. Set one above then change waveform through the range.

## Programs — example 1

The sound of each voice can be easily changed during the playing of a note to give interesting effects.

Try changing the volume during a note. If you set the noise waveform (129) and a long sustain and release and then change the volume slowly up and down, a 'breaking surf' sound can be obtained.

```

100 V=54272:W=129
110 POKEV,25:POKEV+1,25
125 POKEV+5,0:POKEV+6,12*16+15
128 POKEV+4,W
130 FORI=1TO15
140 POKEE54296,I
150 FORT=1TO250:NEXT
160 NEXT
165 REM TURN WAVE OFF AND ALLOW TO DIE
170 POKEV+4,129
185 REM WAIT RANDOMLY FOR NEXT WAVE
190 FORT=1TO10000*RNDC(0):NEXT
195 GOTO129

```

## Programs — example 2

You can also set different ADSR's, waveforms and frequencies for each voice as in this simulated battle. Three voices are used. Voice 3 for plane (waveform 33), Voice 1 for falling bomb and Voice 2 for gunfire.

```

100 V=54272:V2=54279:V3=54286
105 FORI=1TO24:POKEV+I,0:NEXT
106 POKE54296,15
110 POKEV+5,9:POKEV+6,1*16+3:POKEV3+5,
    5*16+15:POKEV3+1,1
120 POKEV+4,17:POKEV3+4,33
130REM PLAY FALLING NOTE
140 FORI=200TO1STEP-1
150 POKEV+1,I:FORT=1TO50:NEXT
160 IFRND(0)<.8THEN180
165 REM SOMETIMES FIRE
170 POKEV2+0,15:POKEV2+1,12:POKEV2+6,
    3*16+3 :POKEV2+4,129:POKEV2+4,128
180 NEXT
185 REM BOMB NOISE
190 POKEV2,10:POKEV2+1,5:POKEV2+6,
    15*16+15 :POKEV2+4,129:POKEV2+4,128
194 REM WAIT FOR NOISE TO DIE AWAY
195 FORT=1TO10000:NEXT
196 REM TURN OFF PLANE AND LET SOUND DIE AWAY
200 POKEV3+4,32

```

There are many different applications of sound. The most common is to mix sound and graphics.

By experimenting with the above programs or modifying programs found in the references, you will rapidly become more expert at developing your own sound routines.

## Demonstration program

This program allows you to play notes on a one octave keyboard and experiment with the ADSR and waveform settings. The function keys and the space bar have the effect of:


Key	Operation
F1	Notes up an octave
F3	Notes down an octave
F5	Set ADSR and wave
F7	End program
space	Change between using only one voice (solo) to using all three voices (poly).

```
100 REM SIMPLE SOUND KEYBOARD
110 DIMN(13),K(13,2):SP$=""
120 V=54272:A(2)=9:A(4)=17:OC=3:PW=4000:Q=0:PF=1:PF$="POLY"
130 FORI=1TO13:READN(I):NEXT:FORI=1TO13:READK(I,0):NEXT
140 FORI=0TO28:POKEV+I,0:NEXT
150 POKE54296,15:GOSUB450:GOSUB630
170 GOSUB600:PRINT"CHANGE ADSR AND WAVE"
180 POKE198,0:SL=7:FORI=0TO4:IFI=4THENSL=14
190 GOSUB610:INPUTA(I):IFI=4AND(A(I)=17ORA(I)=33ORA(I)=65ORA
(I)=129)THEN220
200 IFI<4AND(A(I)>-1AND(A(I)<16)THEN220
210 GOSUB700:SL=SL-1:GOTO190
220 NEXT
230 P=PW:IF A(4)<>65THENP=0
240 FORQ=0TO2:POKEV+3+Q*7,P/256:POKEV+2+Q*7,PAND255
250 POKEV+5+Q*7,A(0)*16+A(1):POKEV+6+Q*7,A(2)*16+A(3):NEXT
260 Q=0:GOSUB450
270 GOSUB600:PRINT"PLAY NOTES"
280 K=PEEK(197):IFK=64THEN280
290 IFK=60THEN750
300 IFK=4THENGOSUB630:GOTO280
310 IFK=5THENGOSUB640:GOTO280
320 IFK=6THEN170
330 IFK=3THENPOKE54296,0:POKE198,0:END
340 N=0:FORI=1TO13:IFK=K(I,0)THENN=I:I=13
350 NEXT
360 IFN=0THEN280
370 IFPFTHENQ=Q+1:IFQ>2THENQ=0
380 IFK(N,1)>255THEN280
390 POKEV+1+Q*7,K(N,1):POKEV+Q*7,K(N,2)
400 POKEV+4+Q*7,A(4)
410 FORT1=1TO50*A(0):NEXT
420 IFPEEK(197)=KTHEN420
430 POKEV+4+Q*7,A(4)-1
440 GOTO280
450 PRINT"[CLR]OCTAVE "OC;TAB(15)"KEYBOARD OR F1,3,5,7"
```

## NOTE

The following lines are difficult to enter


```
460 PRINTTAB(10)"<N>[RVS] [RIGHT] [RIGHT] [-] [RIGHT]
[RIGHT] [RIGHT] [-][OFF]"
```

will look like "  "

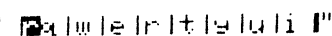
```
470 PRINTTAB(10)"<N>[RVS] [OFF]2[RVS] [OFF]3[RVS] [-],
[OFF]5[RVS] [OFF] 6[RVS] [OFF]7[RVS] [-][OFF]<J>"
```

will look like "  "

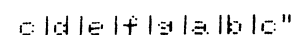
```
480 PRINT"MODE"TAB(10)"<N>[RVS] [-] [-] [-] [-] [-] [-]
[-] <L>"
```

will look like "  "

```
490 PRINTFF$;TAB(10)"<N>[RVS]Q[-]W[-]E[-]R[-]T[-]Y[-]U
[-]I<L>"
```

will look like "  "

```
500 PRINT"[DOWN]NOTE"TAB(10)" OC[-]D[-]E[-]F[-]G[-]A[-]B[-]C"
```

will look like "  "

```
510 PRINT"[DOWN] ADNR VALUES 0 - 15"
```

```
520 PRINT"ATTACK "A(0)
```

```
530 PRINT"DECAY "A(1)
```

```
540 PRINT"SUSTAIN "A(2)
```

```
550 PRINT"RELEASE "A(3)
```

```
560 PRINT"[DOWN]WAVEFORMS TRI SAW PUL NOI"
```

```
570 PRINT"NUMBER 17 33 65 129"
```

```
580 PRINT"WAVEFORM "A(4)
```

```
590 RETURN
```

```
600 SL=19
```

```
610 SL=SL+1:POKE214,SL:PRINT:PRINTTAB(15)SP$:POKE214,SL:
```

```
PRINT:PRINTTAB(15);
```

```
620 RETURN
```

```
630 OC=OC+1:GOTO650
```

```
640 OC=OC-1
```

```
650 OC=OC+(OC>7)-(OC<0)
```

```
660 GOSUB600:PRINT"OCTAVE CHANGE"
```

```
670 FORI=1TO13:F=N(I)*2↑(OC-4)
```

```
680 K(I,1)=INT(F/256):K(I,2)=F-K(I,1)*256:NEXT
```

cont.

```

690 PRINT"[HOME]OCTAVE "00:GOSUB600:RETURN
700 POKEV,6:POKEV+6,0:POKEV+5,8:POKEV+4,17
710 FORJ=100TO20STEP-.5:POKEV+1,J:NEXT:POKEV+4,16:RETURN
720 DATA4479,4745,5027,5326,5643,5979,6334
730 DATA6771,7110,7533,7981,8455,8958
740 DATA 62,59,9,8,14,17,16,22,19,25,24,30,33
750 POKE214,3:PRINT
760 PF=1-PF:PF#="SOLO":IFPFTHENPF#="POLY"
770 PRINTPF#:FORT=1TO100:NEXT
780 GOTO280

```

NOTE	CALC FREQ IN HERTZ	NUMBER	FH	FL
------	-----------------------	--------	----	----

C - 0	16.35	380	1	24
C# - 0	17.32	297	1	41
D - 0	18.35	314	1	58
D# - 0	19.44	333	1	77
E - 0	20.60	353	1	97
F - 0	21.82	374	1	118
F# - 0	23.12	396	1	140
G - 0	24.49	419	1	163
G# - 0	25.95	444	1	188
A - 0	27.49	471	1	215
A# - 0	29.13	499	1	243
B - 0	30.86	528	2	18
C - 1	32.70	560	2	48
C# - 1	34.64	593	2	81
D - 1	36.70	628	2	116
D# - 1	38.89	666	2	154
E - 1	41.20	705	2	193
F - 1	43.65	747	2	235
F# - 1	46.24	792	3	24
G - 1	48.99	839	3	71
G# - 1	51.91	889	3	121
A - 1	54.99	942	3	174
A# - 1	58.27	998	3	230
B - 1	61.73	1057	4	33
C - 2	65.40	1120	4	96
C# - 2	69.29	1186	4	162
D - 2	73.41	1257	4	233
D# - 2	77.78	1332	5	52
E - 2	82.40	1411	5	131
F - 2	87.30	1495	5	215
F# - 2	92.49	1584	6	48
G - 2	97.99	1678	6	142



G# - 2	103.82	1777	6	241
A - 2	110.00	1883	7	91
A# - 2	116.54	1995	7	203
B - 2	123.47	2114	8	66
C - 3	130.81	2239	8	191
C# - 3	138.59	2373	9	69
D - 3	146.83	2514	9	210
D# - 3	155.56	2663	10	103
E - 3	164.81	2822	11	6
F - 3	174.61	2989	11	173
F# - 3	184.99	3167	12	95
G - 3	195.99	3355	13	27
G# - 3	207.65	3555	13	227
A - 3	220.00	3766	14	182
A# - 3	233.08	3990	15	150
B - 3	246.94	4228	16	132
C - 4	261.62	4479	17	127
C# - 4	277.18	4745	18	137
D - 4	293.66	5027	19	163
D# - 4	311.12	5326	20	206
E - 4	329.62	5643	22	11
F - 4	349.22	5979	23	91
F# - 4	369.99	6334	24	190
G - 4	391.99	6711	26	55
G# - 4	415.30	7110	27	198
A - 4	440.00	7533	29	109
A# - 4	466.16	7981	31	45
B - 4	493.88	8455	33	7
C - 5	523.25	8958	34	254
C# - 5	554.36	9491	37	19
D - 5	587.32	10055	39	71
D# - 5	622.25	10653	41	157
E - 5	659.25	11286	44	22
F - 5	698.45	11957	46	181
F# - 5	739.98	12668	49	124
G - 5	783.99	13422	52	110
G# - 5	830.60	14220	55	140
A - 5	880.00	15065	58	217
A# - 5	932.32	15961	62	39
B - 5	987.76	16910	66	14
C - 6	1046.50	17916	69	252
C# - 6	1108.73	18981	74	37
D - 6	1174.65	20110	78	142
D# - 6	1244.50	21305	83	57
E - 6	1318.51	22572	88	44
F - 6	1396.91	23915	93	107
F# - 6	1479.97	25337	98	249

cont.

NOTE	CALC FREQ IN HERTZ	NUMBER	FH	FL
G - 6	1567.98	26843	104	219
G# - 6	1661.21	28439	111	23
A - 6	1760.00	30130	117	178
A# - 6	1864.65	31922	124	178
B - 6	1975.53	33820	132	28
C - 7	2093.00	35831	139	247
C# - 7	2217.46	37962	148	74
D - 7	2349.31	40219	157	27
D# - 7	2489.01	42611	166	115
E - 7	2637.02	45145	176	89
F - 7	2793.82	47829	186	213
F# - 7	2959.95	50673	197	241
G - 7	3135.96	53686	209	182
G# - 7	3322.43	56879	222	47
A - 7	3520.00	60261	235	101
A# - 7	3729.31	63844	249	100
NOT AVAILABLE				

# 18

## Textfiles

### What is a textfile?

A file is a collection of records, which is stored on some medium in a specific order. Each record may contain one or more pieces of data. Each piece of data may be referred to as a **field**.

A telephone directory is a file in which the name, address and telephone number of each subscriber is a separate record. The name, address and telephone number represent different fields. In this case the recording medium is paper.

As files contain a number of data items, this collection of data may also be thought of as a very simple form of a **database**. The database may be used to provide information of various types. It is also possible to **update** the file (ie change, insert or delete data in a record) without affecting any other records.

Using the telephone directory as an example, that database could be used to provide a list of all surnames starting with WI...; a separate list of telephone numbers starting with 268...; or to update any record.

Record 1	Record 2	Record 3
Name	Name	Name
Address	Address	Address
Phone	Phone	Phone

File  
composed of records;  
this file contains 3 records

Field 1	Field 2	Field 3
Name	Address	Phone

Record composed  
of field; Record 1  
contains 3 fields

Char 1	Char 2	Char 3	Char 4
N	A	M	E

Field composed  
of characters; Field 1  
contains 4 characters

Files can also be created, stored and retrieved using the Commodore 64 and a magnetic recording medium.

Already you have been creating, storing and retrieving files using **NEW**, **SAVE** and **LOAD** with respect to the programs you have written.

Let's now look at the commands required to store data, which is not in a program format, on to a magnetic recording medium, ie disk or cassette tape. Data stored in this manner is usually referred to as a **textfile**.

## Two types of textfiles

### Sequential files

This is a textfile in which records, usually of differing lengths, are stored in a serial manner, ie one after the other. A sequential textfile may be used with both disks and cassettes.

One disadvantage of sequential files is that to obtain the data from, say, the tenth record, the previous nine records must be read first. The sequential file can only be read in one direction, ie from the first record through to the last record. Thus, it is not possible to read the tenth record and then go back and read the first record.

Sequential files are ideal for use where the majority of the records have to be read or updated. They are also space saving, in that fields usually do not contain blank spaces.

Record

Field 1					Field 2								Field 3					
N	A	M	E	C/R	A	D	D	R	E	S	S	C/R	P	H	O	N	E	C/R

C/R= character to indicate RETURN

### Relative files

**Note:** The Commodore 64 uses two types of random access file structures, **random** and **relative**. Random files allow true random access to the disk surface. Relative files allow access to specific records within a file. Commodore relative files are similar to random files on other computers and are used in the same manner. Only relative files are covered in this chapter.

Records within a relative file must be of equal length. Each record is given a specific record number. Thus it is possible to read or update records in any order, ie the data in, say, record 10 may be read, then record 2, then record 4, then record 1.

From this it should be obvious that relative files can only be used with disks, and not with cassettes.

Because records must be of equal length, there may be unused space in each record on the disk.

<b>Example:</b>	Field 1	Name	Allow for	15 characters
		Return		1 character
	Field 2	Address	Allow for	25 characters
		Return		1 character
	Field 3	Phone	Allow for	10 characters
		Return		1 character
	Record length			53 characters

However, not all names will be fifteen characters in length, nor will all addresses be twenty-five characters in length, so some space in each record will not be used.

Record	
Char 1	Char 53
↓	↓
Richardson . . . . .	7 Mill St Wynnum . . . . . 268.1234
Wiley . . . . .	13 Baker St Morningside . . . . . 399.2468
McDougall . . . . .	.285 More St Springwood . . . . . 341.5555

## Method of using textfiles

It is important to realise that, when working with a textfile, you will actually be using two files. One file is a program file, which is used to create and access a separate textfile.

The program file will instruct the system to open a textfile; write data to that textfile, read data from that textfile, or update data in that textfile; and finally to close the textfile.

Compare this with the following example of accessing data contained in a filing cabinet.

<i>Filing cabinet</i>	<i>Textfile</i>
Open drawer	Open file
Place data in file	Write data to file
Obtain data from file	Read data from file
Change data in file	Update data in file
Close drawer	Close file

**Remember** — you can't obtain any data until you have opened the drawer/file and, for safety reasons, you must close the drawer/file when you have finished accessing the data.

# Sequential files

## Creating a sequential file — OPEN WRITE PRINT# CLOSE

First — let's create a sequential file (CLASSLIST) and write data to that file.

**Note:** Do not leave a space between PRINT and #. Do not use the abbreviation ?#. This will not work even though it appears correct on the listing.

NEW

```
10 REM PROGRAM NAME CLASSLIST 1.D
15 REM FOR DISK
20 OPEN15,8,15
25 OPEN2,8,2,"0:CLASSLIST,S,W"
30 GOSUB5010
35 PRINT#2,WALBERT"
40 PRINT#2,"GIANNI"
45 PRINT#2,"CARLA"
50 PRINT#2,"LAURIE"
55 PRINT#2,"MONICA"
60 PRINT#2,"EVE"
65 PRINT#2,"RUTH"
70 GOSUB5010
75 CLOSE2
5010 REM DISK ERROR
5020 INPUT#15,EN,EM$,ET,ES
5030 IFEN<20THENRETURN
5040 PRINT"DISK ERROR"
5050 PRINTEN;EM$;ET;ES
5060 CLOSE2;CLOSE15;STOP
```

LIST — check thoroughly

RUN

If you receive an error message, check your program and make the necessary corrections.

If your program is correct, there will be no screen display, as the data has been written to file.

Now save the program:

SAVE "CLASSLIST 1", 8

Now check the directory — there should be two new files, one named CLASSLIST 1 which is described as PRG (for program) and one named CLASSLIST which is described as SEQ (for sequential).

The program CLASSLIST 1 established the following actions:

Line 25 OPEN 2, 8, 2, "0:CLASSLIST, S, W"

OPEN filename      You may select a number between 1 and 128 to indicate the filename (refer channel number below).

Device number      8 = disk drive  
                         1 = cassette  
                         4 = printer  
                         3 = screen

Channel number (or secondary address)      Any number between 2 and 14. It is preferable to also use this number for your filename.

For cassette      1 = WRITE   0 = READ

Drive number      0

Filename      CLASSLIST

Type of file      S = SEQUENTIAL  
                         L = RELATIVE

Direction      Sequential only  
                         W = WRITE   R = READ

Lines 35–65

Data was written to the sequential textfile.

Line 75

Close the filename

The sequential textfile (CLASSLIST) consists of items of data separated by a RETURN character (which you entered at the end of each statement). Each data item (in this case a name) and its following RETURN character, is called a **field**. The fields in this case are of different lengths. The system can recognise the end of each field by the RETURN character.

Do not use fields larger than 80 characters. This is because the INPUT command will not accept more than 80 characters.

## Retrieving data —

**OPEN   READ   INPUT#   CLOSE**

Load and list CLASSLIST 1

Add these statements to your program.

```

200 REM PROGRAM NAME CLASSLIST2.D
210 OPEN2,8,2,"0:CLASSLIST,S,R"
215 GOSUB5010
220 FORC=1TO7STEP1
230 INPUT#2,N$(C)
240 NEXTC
250 CLOSE2

```

LIST — check thoroughly.

RUN

If correct, SAVE "CLASSLIST 2", 8

Line 210 opens the filenumber (2), on disk (8), channel (2), and accesses the filename (CLASSLIST), a sequential textfile (S), and directs READ (R).

Line 220 sets up a loop with seven elements to read the data.

Line 230 inputs the data to memory as an array.

Line 250 closes the filenumber (2).

Don't be dismayed that the names do not appear on the screen when you RUN this program. They are stored in memory! If you wish to see the names displayed on the screen:

LIST CLASSLIST 2

Add these statements:

```

260 FORX=1TO7STEP1
270 PRINTN$(X)
280 NEXTX

```

LIST

RUN

If your program is correct, the names will be displayed on the screen.

SAVE "CLASSLIST 3", 8

## Deleting a textfile SCRATCH

If you wish to delete an entire textfile, first check the directory to obtain the correct filename. Then enter these commands:

```

OPEN 15, 8, 15
PRINT#15,"SCRATCH0:filename"

```

Delete the textfile CLASSLIST.



## Using keyboard to input and retrieve data

Enter these statements, which will provide a program that can be used to create a textfile, enter data in that textfile and then retrieve and print that data.

NEW

```
5 REM PROGRAM NAME INPUT1
6 OPEN15,8,15
10 DIMI$(50)
20 REM T$=FILE NAME, I$=INPUT,C=TOTAL
30 REM H$=HEADING, C$=COLUMN HEADING
40 C=0
50 INPUT"ENTER FILE NAME":T$
60 OPEN2,8,2,"0:"+T$+",S,W"
65 GOSUB5010
70 INPUT"ENTER DATA OR END":I$
80 IF I$="END" THEN120
90 PRINT#2,I$
100 C=C+1
110 GOTO70
120 CLOSE2
130 REM TO RETRIEVE DATA
140 OPEN2,8,2,"0:"+T$+",S,R"
145 GOSUB5010
150 FORK=1TOCSTEP1
160 INPUT#2,I$(K)
170 NEXTK
180 CLOSE2
190 REM TO PRINT DATA
200 PRINT
210 INPUT"HEADING":H$
220 INPUT"COLUMN HEADING":C$
230 PRINT:PRINT:PRINT
240 PRINTH$:PRINT:PRINT
250 PRINTC$:PRINT
260 FORK=1TOCSTEP1
270 PRINTI$(K)
280 NEXTK
5000 CLOSE15:END
5010 REM DISK ERROR
5020 INPUT#15,EN,EM$,ET,ES
5030 IFEN<20 THENRETURN
5040 PRINT"DISK ERROR"
5050 PRINT"EN:EM$:ET:ES"
5060 CLOSE2:CLOSE15:STOP
```

**LIST** — check thoroughly

**SAVE "INPUT 1", 8**

The following exercises will enable you to use the **INPUT** program to create textfiles.

## **Exercise 1**

**LOAD INPUT 1**

**RUN**

For the filename, enter **TEAM**.

For the data, enter eleven (11) surnames, then enter **END** to signify end of data.

For the heading enter **CRICKET TEAM**

For the column heading enter **UNDER 18**

Your output will be a table containing one column.

Check the directory: notice that the textfile has been saved as **T\$**, therefore if you want to use the **INPUT** program again, you will need to allocate a different variable name for the filename.

## **Exercise 2**

**LIST INPUT 1**

Change the variable **T\$** to **L\$** in these statements:

```
20 50 60 140
```

To enable you to output a table with two columns, enter these statements:

```
220 INPUT "ENTER TWO COLUMN HEADINGS";C1$, C2$
```

```
250 PRINT C1$, C2$ :PRINT
```

```
260 FOR K = 1 TO C STEP 2
```

```
270 PRINT I$(K), I$(K+1)
```

**SAVE as INPUT 2**

**RUN**

For the filename, enter **PRICE LIST**.

For the input, enter the name and price for each of five grocery items, then enter **END** to signify end of data.

For the heading, enter **PRICE LIST**.

For the column headings, enter **ITEM** and **PRICE**.

Your output will be a table containing two columns.

Check the directory again; notice that you now have two sequential textfiles each with a variable name for each filename.

## Exercise 3

Now adjust INPUT 2 to create another textfile (Z\$).

Also adjust the column headings input to accept three headings, and the FOR statement to format the output in three columns.

You should now be able to write your own programs to enable you to work with sequential textfiles.

## Relative files

A relative file contains a number of records each of identical length (up to 254 characters). The total number of records available depends on the record size and the free disk space. For example on a new disk you may have 658 records of 254 characters long, or 1316 records of 127 characters long, and so on.

Each record may contain a number of **fields**. These are set in the program and are irrelevant to the record itself.

Each field should be separated by a carriage return  
[CHR\$(13)]

Fields may not contain commas or colons.

The record length equals the sum of the field lengths, plus a return at the end of each field, ie 3 fields of lengths 15, 25 and 10 would require a total record length of  $15 + 1 + 25 + 1 + 10 + 1 = 53$

It is preferable that you do not have more than one relative file open to the disk at one time. It is also preferable not to have a printer turned on when using relative files.

Since the Commodore 64 does not automatically check for disk errors, this must be done after all disk access via the error channel.

Also relative files are controlled using the command/error channel, so you **must open** this channel at the start of a program. (This should be done for all disk access anyway.)

Use OPEN 15, 8, 15

Close this channel last. When you close this channel it closes all disk files in the disk drive only, **not** in the computer.

Relative files are opened and created by:

OPEN A, 8, B, "0:filename, L,"+CHR\$(record size)

where A is the file number (1-128), B is the channel number (or secondary address) (2-14).

Preferably use the same numbers for A and B, eg

OPEN 2, 8, 2, "0:rel test, L,"+CHR\$(53).

Once the file exists, it may be opened by: OPEN 2, 8, 2, "0:rel.test"  
and the disk operating system (DOS) knows that it is a relative file.

**Do not use S, W or R** with relative files.

The record size is set absolutely in the OPEN command. It can **never** be changed during the life of the file.

Any particular record in a relative file may be read or written to at any time. But first you must position the disk to access that record. This is done via the command channel (15), eg:

```
PRINT#15,"P"CHR$(channel no)CHR$(RL)CHR$(RH)CHR$(RP)
from above:
```

Channel number is 2

$RH = \text{INT}(\text{rec.no})/256$

$RL = \text{rec.no} - RH * 256$

RP is a pointer within the record. It is possible to read or write at any place within a record, but it is better to read or write a complete record. Thus use  $RP = 1$ .

If you attempt to write more characters to a record than allowed by the record size, the excess will be lost. Take care when you initially decide on the record size.

When you plan to use relative files, it is preferable (but not necessary) to create all the records you intend to use at the outset. This allows for more efficient disk access and also means you will not run out of disk space later.

The following demonstration program creates a relative file called MAILING LIST and sets up 25 records for use.

The program allows access to any record at any time.

```
100 REM RELATIVE FILE DEMO
110 REM FILE NAME      = T$
120 T$="MAILING LIST"
130 REM RECORD LENGTH = 50
140 REM MAX. RECORDS  = 25
150 REM RECORD IN USE = R
160 REM RH% & RL% = RECORD NO BYTES
170 REM EN,EM$,ET,ES FOR DISK ERROR
180 :
190 OPEN15,8,15
200 OPEN3,8,3,"0:"+T$+",L,"+CHR$(53)
210 GOSUB620
220 INPUT"CREATE NEW FILE (Y/N)";Q$
230 IFQ$<>"Y"ANDQ$<>"N"THEN220
240 IFQ$="N"THEN320
250 :
260 REM CREATE 25 RECORDS
270 FORR=1TO25
280 GOSUB560
290 PRINT#3,"TEST NO"+STR$(R)
300 NEXT
310 :
320 INPUT"READ,WRITE OR END (R,W,E)";A$
330 IFA$<>"R"ANDA$<>"W"ANDA$<>"E"THEN320
340 IFA$="E"THENCLOSE3:CLOSE15:END
350 INPUT"RECORD NUMBER";R
```

```

360 IFR<10RR>25THEN350
370 GOSUB560
380 IFA$="R"THEN480
390 :
400 REM WRITE RECORD
410 INPUT"NAME (15)";N$
420 INPUT"ADDRESS (25)";AD$
430 INPUT"PHONE";P$
440 PRINT#3,N$;CHR$(13);AD$;CHR$(13);P$
450 GOSUB620
460 GOTO320
470 :
480 REM READ RECORD
490 IFEN=50THEN320
500 INPUT#3,N$;PRINTN$
510 IFLEFT$(N$,4)="TEST"THEN320
520 INPUT#3,AD$,P$
530 PRINTAD$;PRINTP$
540 GOTO320
550 :
560 REM SET RECORD
570 RH%=R/256:RL%=R-RH%*256
580 PRINT#15,"P"CHR$(3)CHR$(RL%)CHR$(RH%)CHR$(1)
590 GOSUB620
600 RETURN
610 :
620 REM DISK ERROR
630 INPUT#15,EN,EM$,ET,ES
640 IFEN<20THENRETURN
650 IFEN=510REN=50THENPRINTEM$;RETURN
660 PRINT"DISK ERROR"
670 PRINTEN;EM$;ET;ES
680 CLOSE2:CLOSE15:STOP

```

**Note:** A disk error check is done each time the main disk is accessed.

### Exercises:

- 1 Improve the screen layout.
- 2 Change the file name.
- 3 Change the record size.
- 4 Set up different fields.

# 19

## Word processing

### Introduction

This chapter is to be used with the word processing program **EASYSCRIPT** produced by Commodore Business Machines (UK) Ltd. It is designed as a self-paced instruction program to help you understand the word processing or text editing capabilities of the Commodore 64 when used in conjunction with the **EASYSCRIPT** program.

You will need a blank disk on which to save your documents. This disk will be referred to as the **document disk**.

A glossary of word processing terms is listed in the **EASYSCRIPT** User Guide.

A summary of all commands is provided on a card with **EASYSCRIPT** Master Disk. A copy of this would be helpful to you.

### What is word processing?

Word processing may be defined as the process of preparing, producing and distributing text, eg letters, reports, lists, etc. The term is now applied to the use of specialised equipment for this process.

### Stages of word processing

- 1 Initial input of ideas through dictation, shorthand or longhand.
- 2 Typing, drafting and revision, so that the text is produced in a consistent and clearly readable form. This is commonly referred to as **text editing**.
- 3 Distribution of final copy.

## Starting the system

- 1 Turn equipment on in the following order:
  - a computer
  - b disk drive
  - c monitor or screen.
- 2 Insert EASYSCRIPT disk into drive 0, and close drive door.
- 3 Type LOAD "0:\*",8,1  
Press RETURN.
- 4 The disk motor will run as the program is loaded. (As each block of the program is loaded into the computer's memory, the colour on the screen changes.)
- 5 Initial screen display appears.

When closing system down:

- a remove disk/s
- b use the reverse order to turn equipment off.

## Start-up options

Three questions must be answered before proceeding. Prompts appear on the screen suggesting the most common response. Type the required answer and then press RETURN to move to the next question.

The following options are displayed on the screen:

- 1 Enter text width (40-240) cols?  
Type 40 for the text width in the following exercises.
- 2 Disk or tape?  
Type D if using disk or T if using tape.
- 3 Printer type (0-4)?  
Type a number 0-4 depending on the printer you are using.  
  
0=CBM  
1=MX80  
2=SPINWRITER  
3=QUME/DIABLO/8300  
4=OTHER

To change these start-up options while in EASYSCRIPT press RUN/STOP and RESTORE. Text in memory will not be affected.

The **cursor** will now appear in the top left corner of the screen.

A **status line** will appear above the clear screen.

The status line gives details of:

Mode	Edit	used when inputting or editing a document
------	------	---

Command	used for functions such as printing when changing to this mode the word MODE in the status line will flash
Disk or Tape	used when viewing the directory/index or when deleting a document from disk or tape.
L:001	indicates the line position of the cursor down the screen.
C:001	indicates the column position of the cursor across the screen.

## Formatting a document disk

As your document disk is a totally blank disk it will have to be formatted or initialised. See Chapter 8 of this book for an explanation of this term. Follow the instructions below to format your document disk.

If using one disk drive, remove the EASYSCRIPT Master Disk.

- 1 Insert blank disk into Drive 0 — if using one disk drive.  
Drive 1 — if using two disk drives.
- 2 Enter Disk Mode — press F4 (ie hold SHIFT, press F3)
- 3 Type n0:diskname, -- press RETURN
 

n	command to computer
0	type drive number, either 0 or 1
diskname	up to 16 characters
--	type a 2 character code for identification, eg pn
- 4 Are you sure? is displayed on the screen
- 5 Type Y for Yes. Disk motor will whirr as disk is being formatted.
- 6 00,ok,00,00 will appear on the screen when formatting is complete.
- 7 Press RUN/STOP to return to Edit Mode or press RETURN to remain in Disk Mode to format another disk.

(If formatting is done while creating a document the text in memory is not affected during the formatting process.)

If using one disk drive remove EASYSCRIPT disk and insert formatted document disk. If using two disk drives insert document disk into drive 1. Ensure EASYSCRIPT Master Disk is write-protected.

## Learning about the options

Be sure that you are in Edit Mode (press RUN/STOP to return to Edit Mode).



## Create a new document — title: doc 1

**1** When you commence a new document the system will be in lower case. Use the **SHIFT** key normally in your typing. To type totally in upper case use **LOCK**. Even when locked in upper case, still use the **SHIFT** key to type symbols.

**2** To move the cursor within a document:

<b>RIGHT</b>	<b>CRSR</b> →
<b>LEFT</b>	<b>SHIFT CRSR</b> ←
<b>DOWN</b>	<b>CRSR</b> ↓
<b>UP</b>	<b>SHIFT CRSR</b> ↑

Each key has a repeat function to move the cursor more quickly.

To replace a letter: move the cursor to position and overwrite the correct letter.

**3** To delete a character:

- position cursor one space to right of character
- press **INST/DEL**
- if typing in upper case (**CAPITALS**) release **LOCK** before using **INST/DEL**.

**INST/DEL** will delete the character immediately before the cursor.

Further editing features will be covered in detail later in the chapter.

Do not worry if you have errors in the early documents as you will correct them later, once you have learnt the relevant skills.

**4** If no format instructions are given, **EASYSRIPT** has an assumed format for printing. Format instructions will be covered later in the chapter. The following exercise will use the standard format.

**5** When typing reaches the right side of the screen, the system will automatically place the following letter on the next line. Remember that you have set the text width to 40 spaces. When the document is printed the set margins will be applied. Use **RETURN** only if you wish to return at a specific position, eg end of paragraph. The **RETURN** key will produce a reverse field < on the screen.

*Now type the following, pressing **RETURN** twice to indicate a new paragraph:*

```
Large companies have traditionally used
computers to solve some of their business
problems. Small businesses must also cope
with the increasing complexity of
information, the need for providing
immediate information about their business
and the extremely vital need to control
business costs.<
```

```
<
```

The microcomputer is a system that can be used by the clerk-typist quite easily. It can be placed anywhere an office typewriter would normally be stationed.<

## Saving on disk

The document (doc1) is in the memory of the computer. Should there be a power failure it would be lost. If you wish to save the document for future reference it is essential to save it on disk.

- 1 Press F1 Command Mode
- 2 Press F for File
- 3 Status line now displays "File Name" and cursor blinks.
- 4 Enter document name. Type doc1 RETURN

It is quicker to use lower case for a document name.

**Note:** no spaces and no quotation marks in document name.

Use INST/DEL to make any corrections — do not use other cursor control keys.

- 5 The red IN USE light comes on as the document is being saved.

When the status line displays "Complete" the document is recorded on the disk.

## Create a new document — title: doc2

To clear the memory of the computer:

- 1 Press F1 Command Mode
  - 2 Press E status line displays Erase?
  - 3 Press A indicating **all** data is to be erased from memory
- Screen below status line will clear.

## Comment lines

A comment line contains information about the document, for reference purposes, but is not printed as part of the document.

If the document name is entered in quotation marks it is possible to save the document using a short-cut method.

Each comment line must occupy only one line. If it is to occupy more than one line create a second comment line.

Follow the format below to enter a comment line containing the document name.

- 1 Press F3 — this produces \* (ie inverse asterisk)
- 2 Type    nb“doc2”
- 3 Press RETURN

**Check:** \*nb“doc2”<

*Now type the following:*

Software refers to programs that can be run by a computer. It is generally distributed on media such as disk or cassette for use on various computer systems and is available from your dealer or the manufacturer. Software can also be user-written for specific applications.<

<  
Suppose you want your computer to replace your typewriter and maintain all your personal or business records and accounts. You might choose software for word processing, accounting and file and information management.<

## Short-cut method of saving document on disk — doc2

**Remember:** to use this method the document name must be shown at the beginning of the document in quotation marks in the comment line.

- 1 Press F1     Command Mode
- 2 Press F     File
- 3 Status line displays “File Name”
- 4 Press F2 (ie hold SHIFT and press F1) — document name automatically prints after “File Name”.
- 5 Press RETURN

In future this command will be shortened to F1/F/F2

Do not type the / — it is only intended to separate the functions.

When “Complete” appears in the status line the document is saved.

## Direct cursor moves (non-destructive cursor moves)

It is possible to move the cursor from one position on the screen to another without affecting the memory — this is a **direct cursor move**.

CLR/HOME                      moves the cursor to the top left of the screen

<b>SHIFT-CLR/HOME</b>	moves the cursor to the beginning of the document (useful where document has scrolled off the screen)
<b>F1/G/E RETURN</b>	moves the cursor to the end of the text (Press F1, press G, press E)
<b>F1/G/-- RETURN</b>	moves the cursor to screen line number --
<b>CTRL/W</b>	moves the cursor to the first letter of the following word

*Practise moving the cursor in various directions.*

## Viewing the directory

If you have a document stored in the memory it will not be affected by viewing the directory.

If you wish to see what documents are recorded on disk:

- 1** Press F4      enters Disk Mode (Hold SHIFT, press F3)
- 2** Type \$0      for directory of Drive 0  
     or \$1      for directory of Drive 1
- 3** Press RETURN

Directory on screen will display documents saved on disk.

Doc1

Doc2

The directory also shows how many 'free blocks' remain on the disk.

If the directory has more than 23 lines, the information will scroll off the top of the screen.

To stop display

press SPACE BAR

To continue

press SPACE BAR

To step through directory line by line

press CTRL

### Re-enter edit mode — press RUN/STOP

Any text which was in the computer's memory is then displayed on the screen.

## Loading or retrieving a document from disk

Before being able to edit or print a document you must load the document from the disk into the computer's memory.

- 1** Clear memory and screen (F1/E/A)
- 2** Press F1      Command

**3** Press L status line displays “Load” and cursor blinks

**4** Type document name press RETURN.

*Load doc1 and doc2.*

# Standard or assumed format for printing

Unless you establish commands for printing, EASYSCRIPT will assume the following format:

Page length	66	
Left margin	1	measured from left edge of paper
Right margin	80	measured from left edge of paper
Vertical placement	0	distance from top of page
Text length	60	line on which each page will finish
Single line spacing	0	no blank spaces between lines
Justification	off	
Pitch	10	characters per 2.5 cm

## Create a new document — title: doc3

*Type comment line to include document name.*

**Check:** \*nb“doc3”< (F3 to produce \*)

# Formatting a document

To format the printout of a document means to establish commands for the printer — margins, line spacing, justification, page length, text length, etc.

Remember the following:

- 1** The basic command \* (F3) must appear on the left margin at the beginning of the line where the format is listed.
- 2** Format instructions must be separated by a colon (:).
- 3** The format line must be followed by a RETURN(<).
- 4** It is necessary to change only those commands which are different from the standard format, but in the format instructions for doc3 all abbreviations are shown.
- 5** The format instructions will apply to the text following until the format instructions are changed.

*Set the following format for doc3. (Note the abbreviations used.)*

Left margin 10; right margin 70; double line spacing; length of paper 66; printing to end 60; justification on; top margin 10; pitch 12

### Check format instructions:

\*1m10:rm70:sp1:pl66:tl60:ju1:vp10:pt12<

Points to note: Standard format is single line spacing — sp0

sp1 — indicates one blank space between lines, ie double line spacing

sp2 — indicates two blank lines, ie treble line spacing

ju1 — justification is turned on

ju0 — justification is turned off (standard format)

### To change format

Position cursor over instruction to be changed and retype instruction.

### To centre heading

This instruction should be on a separate line:

\*cn1;HEADING\*cn0<

If centring more than one line, type headings to be centred and then turn centring off:

\*cn1;HEADING<

<

DATE\*cn0<

### To print text flush right: text will finish on right margin

This instruction should be on a separate line

\*ra1;TYPE TEXT\*ra0<

*Centre these headings:*

WIN THE PAPER WAR

Your name

*Flush right:*

Current date

**Check:** \*cn1;WIN THE PAPER WAR<

<

Your name\*cn0<

\*ra1;current date\*ra0<

*Now type the following document (return to lower case):*

Many small businesses today are staggering under an avalanche of paperwork. With spiralling wages it is harder and harder to keep costs down while managing the necessary workload. New technology has brought computers into the reach of most small businesses allowing them to reap the

benefits previously afforded only to large organisations.<

<

The use of pre-packaged software is a large saving to small businesses because the cost of writing the program is shared among the many people who use it. As the programs have been previously tested, valuable time is saved.<

*Save this document on disk.*

**F1/F/F2 RETURN** — if using Comment Line

**F1/F/doc3** — no Comment Line

*Print a copy of the document.*

Insert paper in printer and turn printer ON.

**1 Press F1** Command Mode

**2 Press O** status line displays "Output"

**3 Press P** for Printer — the printer assumes you are using separate sheets — press C to print next page.

Abbreviated command **F1/O/P**.

If you have a format error, an error message will appear. Check the EASYSCRIPT User Guide for an explanation of all error messages.

## **To print multiple copies**

**1 Press F1**

**2 Press O**

**3 Press X** status line displays "Number of Times"  
Type number of copies required, eg 2  
Press RETURN

**4 Press P**

Printing will stop at the end of each page.

Press C to continue printing.

*Print a copy of doc3.*

## **Videoprint**

As the format instructions for printing are different from the width of text displayed on the screen it is possible to do a videoprint.

This means that it is possible to view the document in the format in which it will appear on the paper.

**1 Press F1** — Command Mode

**2 Press O** — status line displays "Output"

**3** Press V — for Video (or screen)

The screen acts as a 'window' for 40 spaces across the screen and 23 lines down the screen. Obviously all the text cannot appear on the screen. Any lines commencing with \* will not appear in the videoprint.

## Scrolling — to move cursor

Right	CRSR →
Left	SHIFT CRSR ←
Down	C =
F7	Tabs across 20 spaces
F5	Tabs across 40 spaces

Tap SPACEBAR to start or stop scrolling.

You cannot move cursor back up — you must press RUN/STOP and start again.

Press RUN/STOP to return to Edit Mode.

### Load doc1 from disk

Clear the memory and load the document.

This document is now in the memory of the computer and on the screen. You will notice that the text scrolls up the screen. Although the text has scrolled from the screen, it is still in the memory of the computer.

Move the cursor to the end of the document.

Press RETURN twice to allow a blank line between paragraphs.

*Make the following addition to doc1:*

```
The microcomputer brings a new age to
office automation. Now, the small
businessman does not have to make his
office fit the traditional computer system,
ie a separate central processor, mass
storage device, printer or video display
unit, or purchase expensive furniture to
house this type of system.<
<
```

```
The integration of hardware and software
offers an outstanding total computer system
for the small businessman.<
```

**Note:** What you have on disk (doc1) is now different from what is in the memory of the computer.



## Save the amended document — using the same title: doc1

This is an updated version of the previous doc1 and will be saved under the same name.

As you did not use a comment line in doc1 follow the normal procedure.

**1** F1/F/doc1 because you have already saved a document under this title, the computer will display REPLACE FILE?

**2** Type Y for Yes

The computer will now save this as the new doc1.

*Load doc3.*

*Make the following addition to doc3:*

*Centre the heading: DON'T BE A LOSER*

**Check:** \*cn1;DON'T BE A LOSER\*cn0<

Your business could be losing money through inefficient collection procedures. Overdue accounts tie up valuable working capital and if not followed up can result in bad debts.

Change format for the following text — left margin 20, right margin 60, single line spacing, justification off.

\*lm20:rm60:sp0:ju0<

The reports in this program can save time and money by showing you which accounts are overdue and how long they have been outstanding. You can write your own standard collection letters which will be produced by the computer and will incorporate all relevant information.<  
<

At the end of the month statements can be generated automatically thus saving time and costly mistakes.<

## Save this document as a separate document, ie using a different document name

If using a comment line — move cursor to beginning of document  
— move cursor to comment line and change document name to doc4 by overtyping  
— F1/F/F2

No comment line — F1/F/doc4

You now have these documents recorded separately — doc3 and doc4.

*Print doc4.*

Compare the printed output of doc3 and doc4 and note the additional information in doc4.

## Printing the directory

If you wish to print a copy of the directory you will lose anything stored in the memory. Be sure you have saved it on disk.

**Remember:** if you view the directory on the screen, it does not affect the document in the memory.

- 1 Clear screen (F1/E/A)
- 2 Press F4 — Disk Mode
- 3 Type +\$0 or +\$1 if using drive 1 RETURN
- 4 Print in usual way (F1/O/P)

When printing is completed the computer automatically returns to Edit Mode.

## Editing

To make changes to a document you must firstly load it from the disk into the memory.

*Load doc3.*

### Insertions

*One character or several characters*

- 1 Position the cursor where you wish to insert the data.
- 2 Hold SHIFT and press INST/DEL.
- 3 The text will move to the right and will allow a blank space. If several characters are to be inserted repeat the process — further spaces will be opened.
- 4 Type the insertion (be careful not to use any of the cursor control keys while in Insert Mode).

*More than several characters*

- 1 Position the cursor where you wish to insert the data.
- 2 Press F1.
- 3 Press I — status line displays “Insert ON”.
- 4 Type the insertion (including space after last letter or two spaces after end of sentence).  
The original text moves to the right to allow the insertion.
- 5 When the insertion is completed turn off the Insert Mode.  
F1/I — status line will display “Insert OFF”.

*Additional RETURN/S*

Position cursor where additional linespace is required — press RETURN.

*Make the following insertions to doc3:*

<sup>et</sup>  
 Many small businesses today are staggering  
 under an avalanche of paperwork. <sup>Is your business</sup>  
<sup>With in this category?</sup>  
 spiralling wages it is harder and harder to <sup>New para</sup>  
 keep <sup>these</sup> costs down while managing the  
 necessary workload. <sup>Correct please</sup>

## Deletions

### One character

- 1 Position the cursor one space to the right of the letter to be deleted.
- 2 Press INST/DEL — the character to the left of the cursor is deleted.

### Word/s

- 1 Position the cursor over the first letter of the word/s to be deleted.
- 2 Press F1.
- 3 Press D — status line displays “Delete”.
- 4 Press CSR→until the word/s in the following space are highlighted (in reverse image).
- 5 Press RETURN — the highlighted text will be deleted and the text closes up.

### One line

- 1 Position the cursor over the first letter of the section to be deleted.
- 2 F1/D
- 3 Press CSR ↓ — the line will be highlighted (in reverse image).  
The cursor moves down one line.  
If further lines are to be deleted press CSR ↓.
- 4 Press RETURN — the highlighted text will be deleted and the text will reformat.

## Erasing text

Erasing text is different from deleting text. Characters are erased but the text does not reformat, ie blank spaces are left where the characters/words were erased.

To erase      sentence

Position the cursor at the first letter of the sentence.

The computer will erase to the next full stop.

It does not recognise ? or ! as the end of a sentence.

To erase	<i>paragraph</i>	Position the cursor at the first letter of the paragraph.  Computer will erase from cursor to next <.
To erase	<i>remainder</i>	The computer will erase from the cursor to the end of the document.
To erase	<i>all</i>	It does not matter where the cursor is positioned — all text is erased.

*Follow these steps:*

- 1 Press F1
- 2 Press E
- 3 Type S      Sentence  
          P      Paragraph  
          R      Remainder  
          A      All text

*Make the following corrections to doc3:*

Many smaller businesses today are staggering under an avalanche of paperwork. <sup>delete</sup>

Is your business in this category? With spiralling wages it is harder <sup>delete</sup> and harder to keep costs down while managing the <sup>ever increasing</sup> ~~necessary~~ workload. New technology has brought computers into the reach of most small businesses allowing them to reap the benefits previously afforded <sup>to</sup> only ~~the~~ large <sup>an immense</sup> ~~large~~ organisations.

← <sup>add extra return</sup> The use of pre-packaged software is ~~a large~~ saving to small businesses because the cost of writing the program is shared among the

~~many~~ <sup>users</sup> people who use it. As the program ~~has~~ <sup>has</sup> been previously ~~tested~~ <sup>trialled</sup>, valuable time is saved.

**Remember:** what you have on the screen is now different from what is stored on disk.

### Save the new document — use the same name: doc3

*Load doc1.*

- 1 Insert a comment line to show document name.
- 2 Change the format from a standard format by inserting the following format instructions:

left margin 20, right margin 60, top margin 15, double line spacing, text justified

\*lm20:rm60:vp15:sp1:ju1<

- 3 It is possible to leave blank lines in a document by inserting a format command:

\*ln6<

The number indicates the number of blank lines before printing recommences.

The command should be positioned at the beginning of a line. This takes less memory space than using 6 returns.

*Insert command for 10 blank lines between paragraphs 2 and 3.*

- 4 To emphasise text — underscore or use bold print. These commands will vary between printers.

Position cursor where emphasis is to commence F1 [

Position cursor where emphasis is to finish F1 ]

*Select two words and emphasise them.*

- 5 To change UPPER CASE to lower case, or reverse.

Position cursor over first character to be reversed.

F1/U — **all** text following cursor will be changed.

Position cursor one space to the right of where you wish to finish the change, eg end of word/s, sentence.

F1/U — **all** text following the cursor will revert to previous style, leaving only the desired section of text altered.

*Select several words or a sentence and change to upper case (capitals).*

## Printing a document

Continuous stationery      F1/O/C

Individual sheets      F1/O/P      when first page is completed  
insert next sheet and press C to  
print next page.

**Note:** The printer will automatically stop printing at line 60 and will print the remainder on the second page.

**Save the amended document using the same name**

## Scanning the directory to load a document

If you are unsure of a document name it is possible to scan the directory until you find the document required and then to load.

- 1 Clear screen.
- 2 Press F4 — Disk Mode
- 3 Type +\$0 and press RETURN (type +\$1 if using Drive 1).  
This loads the directory and returns to Edit Mode.
- 4 To select files F1/L/F2.

The disk name appears on the status line.

Each time you press F2 the next document name is shown in the status line.

When the document name you wish to load appears, press RETURN.

The document is then loaded into the memory and appears on the screen.

### Scan the directory and load doc 1

*Load doc4*

## Viewing a document quickly

F1/SPACEBAR      display next 23 lines of text

F1/SHIFT-SPACEBAR      display previous 23 lines of text

Practise these two features.

*Make the following corrections to doc4:*

Your business could be losing money through  
inefficient collection procedures. <sup>Any</sup> ~~Overdue~~  
accounts tie up valuable working capital  
and if not followed up <sup>by you</sup> ~~can~~ result in bad

*Use format command to insert 6 blank lines*

debts/ *being* incurred.

The reports in ~~this~~ <sup>these</sup> program<sup>s</sup> can save <sup>both</sup> time and money by showing you which accounts are overdue and how long they have been outstanding. You can write your own standard collection letter <sup>and this</sup> ~~which~~ will be produced by the computer <sup>automatically</sup> and will incorporate all relevant information. At the end of the month statements can be generated automatically thus saving time and costly mistakes. <sup>delete</sup>

### Save the amended document on disk

Print documents doc4 doc3.

(Note the amendments to these documents.)

## Deleting or scratching a document from disk

- 1 Press F4 — Disk Mode
- 2 Type S0:docname — RETURN
- 3 Are you sure? Type N to stop deletion  
Type Y to delete/scratch document
- 4 Message appears to show document has been deleted.  
01,filescratched,01,00

Delete doc2 from disk.

When complete, list directory to check deletion.

### Create a new document — title: train1

- 1 Create a comment line to show document name.
- 2 Create a comment line — using tab to indent paragraphs.
- 3 Use left margin 20; right margin 60; and double line spacing

## Setting tab stops

- 1 Move the cursor to position of tab stop (use column position in status line).

- 2 Press F1.
- 3 Press T — status line will display “Set Tab”.
- 4 Press H — to indicate horizontal tab.

If setting more than one tab, repeat the above process for each.

## To move to tab position

Press F7.

## To display tab positions

- 1 Press F1.
- 2 Press P — print tabs.

Tab stops are indicated on the status line by a /

*Type the following:*

The increasing use of computer technology not only influences the way that businesses and organisations conduct their affairs, but also the way we live. The impact on people has been dramatic.<

<

A critical issue is whether or not the total number of jobs is increased or decreased by this technological innovation.<

<

The basic structure of employment has changed as many existing jobs have altered or been eliminated.<

<

New jobs have been created in computer-allied industries, such as retail outlets for computers as well as technicians to service machines.<

## Save this document on disk — train 1

*Print document.*

## Ranging

Ranging simply means to highlight or define a block of text to be copied or transferred.

- 1 Position the cursor over the first character in the block.



- 2** F1/R — status line displays “Set Range”.
- 3** Move the cursor right and/or down over the text to be highlighted. (If you move too far simply move cursor up or left to remove highlighting.)
- 4** Press RETURN.

This block of text is stored in the memory of the computer.  
 To cancel a block during set-up, move the cursor back to the first character in the block and press RETURN.  
 The range or block will remain in the memory until a further range is created.

*To move the position of a word or block within a document:*

- 1** Range (highlight) a block of text. This may include any length of text, eg one word, a sentence, a line or a paragraph.
- 2** Position the cursor where the block is to start.
- 3** Press F1.
- 4** Press X.

The text to the right of the cursor moves down to allow insertion of the block. The block is deleted from its original position.

*Practise moving words, lines, paragraphs or blocks of text.*

## Save/call areas

It is possible to define a section of text and call it up elsewhere in the document, ie to repeat it.

- 1** Range or highlight a block of text to be duplicated.
- 2** Position the cursor where the block is to be repeated.
- 3** F1/A the block still appears in its original position but is duplicated at the cursor position.

This block can be duplicated as many times as you wish.  
 The ranged text will be stored in the memory until a further range is created.

*Define paragraph 1 and call it up between paragraphs 2 and 3 and also at the end of the document.*

*Print the final copy of the document.*

## Using the tabulator

To set tab stops	Move the cursor across the screen to tab position F1/T/H To set further tabs repeat the process
To move to tab position	F7
To display tab settings	F1/P Tab position is indicated by / in status line

- To clear one tab stop      Position the cursor at the tab position to be cleared  
    F1/C/H  
    To check that tab is cleared, display tab settings
- To clear all tab stops      F1/Z/H  
    Status line will display “Zero All Tabs” and the cursor will blink

Tab settings are retained in memory until:

- a computer is switched off
- b all tabs are cleared
- c another document is loaded with tabs saved.

This is helpful if you are creating several documents which require the same tab settings.

**Create a new document — title: tabl**

- 1 Set margins to allow a 40 space line — eg \*lm20:rm60<
- 2 Centre this heading — AUSTRALIAN CITIES.
- 3 Insert two blank lines after the heading.

*To calculate tab positions on the screen:*

Queensland*	Brisbane	Cairns
Victoria	Melbourne*	Geelong
Tasmania	Hobart	Launceston*

- a Number of characters on line — less one space to allow for < (preset line length — 40 characters) 39
- b Add longest entry in each column (indicated by \*)  
     (10+9+10) 29
- c Blank spaces 10
- d Distribute blank spaces evenly between columns (divide blank spaces by 2 in this exercise) 5

*To determine tab positions:*

- 1st column — starts at 1st character position on screen
- 2nd column — starts at (10+5) 15th character position on screen
- 3rd column — starts at (15+9+5) 29th character position on screen

**Remember:** To finish each line that has tabs press RETURN.

- 4 Move the cursor to the left of the screen.
- 5 Type table (do not type \*).

## To save tabs with a document

Add a + sign to the end of the document name.

F1/F/tab1 +

When reloading this document remember to include + in the document name.

*Print a copy of tab1.*

*Make the following additions to tab1:*

**1** Change the body of the table to double line spacing.

**2** Clear the tab stop at 15.

Type the following:

New South Wales	Sydney
South Australia	Adelaide

**3** Set a tab stop at 22nd character position on the screen.

Type the following:

Western Australia	6000	Perth
Northern Territory	5790	Darwin
ACT		Canberra

*Save the new document — tab2 — add + to document name to save tab settings.*

*Print tab2.*

## Widening text screen

**Create a new document — title: tab3**

Clear all tab settings — this is a safeguard as previous tabs are retained in the memory.

If you choose a line length which is wider than the screen width, ie 60 spaces, the screen acts as a 'window' for portion of the text.

As you type, the document will scroll across the screen.

## Changing video/screen width

Return to start-up options — press RUN/STOP and RESTORE.

*Change screen width to 80 spaces.*

**1** Use comment line to show the document name.

(tab3+ if you wish to save tab settings.)

**2** Set left margin 10; right margin 90.

**3** Calculate the tab positions for the following table.

**4** Set tab positions.

**5** Type table.

## FOREIGN LANGUAGES AND COMMUNICATIONS

Instructor	Budget No	Class Name	Teaching Units
H. Jenkins	1230-3	French	3
P. Rackermann	1970-2	Japanese	4
B. Smith	1345-2	German	3
F. Gillespie	1911-4	Spanish	3
R. Dillon	1238-2	Italian	4

**(Check:** Possible 79; use 13, 9, 10, 14; 3 spaces between columns.)

Save the document — tab3+. Be sure to save the tab settings.

Print tab3+

(If you wish to reload this document at a future time you will have to readjust the screen width to 80 spaces.)

## Decimal tabs

Clear all previous tab settings.

A decimal tab is set to allow easy alignment of decimal points in columns of numbers.

The tab position is set at the point where the decimal is to be positioned — always allow for the longest whole number when calculating tab positions.

When using a decimal tab stop, any characters typed are moved to the left of the tab position until the decimal point is entered and then typing carries on as normal.

*To set decimal tab position:*

- 1 Press F6 — status line displays “Decimal On”.
- 2 Set the tab in the normal manner (F1/T/H).
- 3 Press F6 — to turn off the decimal indicator.

Decimal tabs are indicated by # in the status lines when tab positions are displayed.

When a decimal tab position is reached in an exercise, the status line displays D.

## Create a new document — title: tab4

- 1 Clear all previous tab settings.
- 2 Change screen width to 60 spaces.
- 3 Set margins to allow for a 60 space line.

*Calculate decimal tab positions and type the following:*

## ANNUAL EXPENSES

	Budget	Expended	Balance
Telephone	250.00	230.85	19.15
Parking fees	75.00	68.00	7.00
Administrative fees	1325.00	1230.00	95.00
Publicity	1100.00	1000.00	100.00

**Remember:**    Possible spaces  
                    Number of spaces used  
                    Spare spaces left  
                    Divide spare spaces by? (3)  
                    Calculate starting point of columns  
                    Calculate decimal tab positions  
                    Set decimal and normal tab stops.

## Renaming a document

- 1 Clear memory/screen
- 2 Press F4 — Disk Mode
- 3 r0:newname=oldname    RETURN

*Rename doc1 as sample.*

**Check:** r0:sample=doc1    RETURN

*Check directory to see changed document name.*

## Create a new document — title: train3

*Type the following:*

In an effort to ensure award coverage for all public sector professional engineers in Tasmania the Association has commenced proceedings to provide award coverage for the professional engineers employed by the Port Authority Tasmania and Transport Commission, Tasmania.<

<

This will then give the engineers concerned the opportunity to take matters such as reclassification to the Arbitrator in Tasmania.<

## Searching and replacing

Sometimes you may wish to search for a particular word or group of words in a document and substitute another word in its place (eg month, surname, town).

- 1 Position the cursor at the beginning of the text.  
(SHIFT CLR/HOME)
- 2 Press F1.
- 3 Press S — status line displays “Search”.
- 4 Type word/s with a space before and after.  
(If searching for ‘the’ and no space is included before and after, the computer will select all ‘the’ combinations, even in the middle of a word, eg ‘father’.)  
Type engineers — press RETURN.
- 5 Status line displays “Replace”.  
Type the word with a space before and after.  
Type surveyors — press RETURN.

### To begin searching and replacing

- 1 Position the cursor at the beginning of the text.  
(SHIFT CLR/HOME)
- 2 Press F1.
- 3 Press @.
- 4 Press M — to indicate that the document is in the memory.

(It is possible to search and replace words/phrases in “Linked Documents”, ie documents which are to be joined — F1/@/L)

*Practise this by making the following substitutions in train2.*

The cursor must be at the beginning of the text before starting.

- 1 Replace “ award coverage ” with “ correct salaries ”.
- 2 Replace “ Tasmania ” with “ Queensland ”.

## Hunting

Sometimes you may wish to search/locate/hunt for a particular word or group of words within a document.

- 1 Position the cursor at the beginning of the text.
- 2 F1/S status line will display “Search”.
- 3 Type word/s to be located.  
Press RETURN.  
Status line will display “Replace”.
- 4 Press RETURN to indicate no replacement.

### To begin hunting

F1/H/M

Computer will locate word/s. To locate further examples repeat the process.

*Hunt for “ public sector ”.*

## Standard paragraphs

1 Use a 50 space line — select your own margins.

*Create each of the following standard paragraphs as a separate document and save on disk.*

**Remember** — Clear memory before creating each new document.

Title for each document — para1 para2 para3.

para1

We wish to bring to your notice the fact that the above account is now overdue. We feel sure that this must be an oversight on your part and look forward to receiving a cheque within the next few days.<

para2

May we draw your attention to the fact that the above account is now overdue. Settlement by return mail would be appreciated.<

para3

As we have received no reply to our previous letters regarding this account, we regret to advise that the matter has now been placed in the hands of our solicitors, Messrs Jones & Associates, and you will be liable for the costs incurred.<

## Using standard paragraphs to create a new document

It is possible to merge several documents.

This is done by loading the document from the disk and chaining to the end of the text on the screen.

### Create a new document — title: insert

*Type the following paragraph:*

An important task required in word processing is combining standard paragraphs to make a new document.

*Follow these steps:*

- 1 Position the cursor where the new document is to be loaded, ie press RETURN twice.
- 2 F1/I — status line will display “Insert ON”.
- 3 Load para2 (F1/L/para2 RETURN)
- 4 Turn off Insert Mode. (F1/I)
- 5 The paragraph will chain to the end of the previous paragraph.
- 6 Move the cursor to the end of para2.
- 7 Press RETURN to allow a blank line between paragraphs.
- 8 Type the following:  
**Each standard paragraph should be composed as a separate document.**
- 9 Press RETURN twice.
- 10 Call up para3.
- 11 Move the cursor to the end of the text.
- 12 Call up para1.

*Print a copy of the document.*

*Save this document on disk — title: insert.*

### **Create a new document**

As we do not intend to save this form letter there is no need for a document name.

We are going to create a form letter using the standard paragraphs created in the previous exercise.

- 1 Enter your format commands.
- 2 Use full block style of letter with open punctuation.
- 3 Use the current date.
- 4 Send the form letter to Mr K King  
15 Cross Road  
YERONGA QLD 4104
- 5 Use a salutation: Dear Sir  
Press RETURN twice to allow a blank line.
- 6 Use para1 for the body of the letter.
- 7 Finish the letter with a complimentary close: Yours faithfully

Manager

*Print a copy of this letter — take a file copy.*

### **Create new document — title: sample1**

*Create the following comment lines:*      document name  
sample header and footer



# Headers and footers

Sometimes you may wish the same information to be printed at the top (head) or bottom (foot) of each page.

These are referred to as headers and footers.

In this document we will:

provide a header which should be centred      **SAMPLE HEADER**

provide a footer which should be right

justified (flush right/pivoted)

current date

(eg 25 June 19—)

The format instructions must be placed at the start of the document, following the comment lines.

The header and footer instructions must be on a separate line.

It is possible to have up to three items on the same line in a header or footer — they should be separated by a comma.

The text before the first comma is printed on the left margin.

The text between the two commas is centred.

The text following the second comma is right justified.

NAME, SAMPLE HEADING, DATE

NAME,, DATE

SAMPLE HEADING, DATE

## Header

\*hd2:,,SAMPLE HEADER,<

\*    Press F3

hd    Header command

2    Leave 2 blank lines after the header before starting the text

,SAMPLE HEADER,    Indicates that header is centred.

<    Press RETURN

## Footer

\*ft3:,,CURRENT DATE<

\*            F3

ft            Footer command

3            Leave 3 blank lines from the end of the text to the footer

„CURRENT DATE            Right justified

The footer will finish on Text Length (tl) command position — in a standard format this is on line 60.

## Indicating a new page (ie forced page)

To indicate that a new page is required even though the page is not full:

\*fp0<

This may be on a separate line or may appear at the end of a paragraph.

*Press RETURN and type the following exercise so that each section will appear on a new page:*

This is the first page of the sample document to print a header and footer to include centring and flush right.<

<

There are many reasons why some small businesses fail while others go on to earn good profits and perhaps grow into medium-size or even larger firms.<

\*fp0<

*Type the following:*

Failure to make the right decisions is certainly an important contribution to lack of business success. If you select a declining rather than a growing industry, you can make the battle for sales and profitability especially difficult.<

\*fp0<

*Type the following:*

Many small businesses fail simply through poor management: failure to do an adequate job of planning and controlling their activities, organising themselves or their staff effectively, choosing employees wisely and leading them well.<

<

The successful small businessman performs these functions effectively.<

*Print document — sample 1. (Press C to print next page)*

*Save document on disk — title: sample 1.*

## Automatic page numbering

It is possible to number each page automatically. Page numbering will start from 1.

The page number may be included in a header or a footer.

To indicate automatic page numbering within a header or footer:

Press F1

Type # (ie Shift-3)

The # will be reverse field.

## Create a new document — title: sample2

- 1 Insert two comment lines      name of document  
AUTOMATIC PAGE NUMBERING
- 2 Enter a header      automatic page numbering (right justified)
- 3 Enter a footer      page # (centred)
- 4 Type the same three pages from the previous document, ie sample1.

### Check:

```
*hd2:,,AUTOMATIC PAGE NUMBERING<
```

```
*ft3:,,PAGE#,<
```

*To start page numbering at a different number:*

Before entering header or footer command, type:

```
*P#--
```

The # will not be in reverse field in this command line. The number (--) will indicate the first page number to be used.

*Enter header and footer commands as shown previously with # in reverse field.*

*Print document.*

*Save document — sample2.*

## Additional functions

### Videoprint

In a longer document it is possible to view the document page by page.

Output to video.

To indicate that the document is continued over the page, a small row of squares appears along the bottom of the screen.

To display the next page — press C (Continue).

To indicate the end of the document a solid bar appears below the end of text.

## To print selected pages

In a long document you may wish to print or reprint only selected pages.

Output to video.

Move through the text on the screen until you reach the page you wish to print. Position the cursor at the end of the previous page.

- 1** Press P — next page will be printed.
- 2** Press C — following page will be printed.
- or** Press V — output will return to video.

## Conclusion

The main functions available in the EASYSCRIPT word processing system have been covered. You should now be able to apply the basic principles to use any further functions with which you wish to experiment.

# Index

addition, 15  
arrays, 81

BASIC, 4  
byte, 2

C=, 7  
cassette recorder, 4, 41  
cassettes, 4  
centring headings, 78, 136  
changing statement, 28  
changing data, 23  
character generator, 95  
clearing data, 24  
clearing memory, 29  
clearing screen, 29  
CLOSE, 36, 120  
CLR, 8  
colon, 13  
colour, 88  
comma, 13  
comment line, 132  
conditional statement, 50  
CONT, 49  
CONTROL, 7  
COPY, 38  
CPU, 2  
create, 131  
cubing, 16  
cursor, 5, 7

DATA, 45  
decimals, 17  
decimal tabs, 150  
DEL, 8  
delete, 9, 28, 39, 145  
DIM, 84  
direct cursor moves, 131, 133  
directory, 37, 134  
disk drives, 4, 34  
disks, 4, 34  
display, 3, 13  
division, 16  
dummy data, 72

edit, 9, 28, 134, 140  
end, 27  
erase, 141  
exponentiation, 15

firmware, 2  
flowchart, 60  
flush right, 136  
footer, 155  
FOR, 52  
format, 130, 135

GOTO, 48  
graphics characters, 87  
graphics mode, 89

hardware, 2  
headers, 155  
high resolution graphics, 99  
hunt, 152

IF . . . THEN, 50  
INPUT, 43  
INPUT#, 121  
INST, 8, 9  
INT, 18, 73  
integer variables, 24

keyboard, 3  
kilobyte, 2

languages, 4  
LEN, 72  
LEFT\$, 74  
LET, 22  
LIST, 27  
LOAD from cassette, 42  
LOAD from disk, 38  
logical operators, 18  
loops — multiple, 56  
loops — nested, 57, 83

mathematical functions, 18  
memory, 2  
memory address, 22  
memory — screen and colour, 93  
memory size, 2

MID\$, 74  
 modes — upper/graphic, 6  
 modes — upper/lowercase, 6  
 multiple copies, 137  
 multiplication, 16  
  
 NEW, 27  
 new page, 156  
 numeric data, 12  
 numeric variables, 24  
 NEXT, 52  
  
 ON . . . GOTO, 72  
 ON . . . GOSUB, 73  
 OPEN, 120  
 order of precedence, 20  
  
 page numbering, 156  
 PCM, 95  
 pigeon-holes, 22  
 pixel, 95  
 POKE, 88  
 PRINT, 11  
 PRINT#, 36, 120  
 PRINT abbreviated, 12  
 PRINT multiple, 13  
 PRINT statement length, 13  
 printer, 4, 31  
  
 question mark, 12  
  
 RAM, 2  
 RANDOM, 18  
 ranging, 146  
 READ, 45, 121  
 real variables, 24  
 relational operators, 18  
 relative file, 125  
 REMark, 27  
 RENAME, 39, 151  
 reorganise disk, 39  
 replace, 37  
 retrieve, 23  
 RESTORE, 7, 71  
 RIGHT\$, 74  
 ROM, 2  
 rounding, 17  
  
 RUN, 27  
 RUN/STOP, 8  
 RVS, 8  
  
 SAVE, 132  
 Save/Call, 147  
 SAVE on cassette, 41  
 SAVE on disk, 37  
 scientific notation, 17  
 SCRATCH, 122, 145  
 scrolling, 138  
 search and replace, 151  
 semicolon, 13  
 sequential file, 120  
 software, 2  
 sprites, 86, 99  
 square root, 17  
 squaring, 16  
 standard format, 135  
 statement numbers, 27  
 statement — conditional, 50  
 statement — unconditional, 48  
 status line, 129  
 storing data, 22  
 string data, 12  
 string variables, 24  
 subroutine, 71  
 subtraction, 16  
 syntax error, 8, 12  
  
 TAB, 13  
 tabulator, 145, 147  
 text mode, 5  
  
 unconditional statement, 50  
  
 VAL, 74  
 variables, 24  
 variables — numeric, 24  
 variables — string, 24  
 VERIFY on cassette, 41  
 VERIFY on disk, 38  
 videoprint, 137, 157  
  
 WRITE, 120  
 write-permit notch, 35  
 write-protect tab, 35





---

*Shake hands with the Commodore 64* has been designed as an individualised programme to help you learn the operation of the Commodore 64 microcomputer.

Operating instructions relate specifically to the Commodore 64, but some of the instructions may also apply to other brands of microcomputers.

*Shake hands with the Commodore 64* is written in an easy-to-understand style which assumes no knowledge of computers or computing jargon. It takes you from starting the system to simple programs and programming, then to graphics and finally to word processing.

---